Contents

Chapter 1

Advantages of BST over Hash Table

Advantages of BST over Hash Table - GeeksforGeeks

Hash Table supports following operations in $\Theta(1)$ time.
1) Search
2) Insert
3) Delete

The time complexity of above operations in a self-balancing Binary Search Tree (BST) (like Red-Black Tree, AVL Tree, Splay Tree, etc) is O(Logn).

So Hash Table seems to beating BST in all common operations. When should we prefer BST over Hash Tables, what are advantages. Following are some important points in favor of BSTs.

1. We can get all keys in sorted order by just doing Inorder Traversal of BST. This is not a natural operation in Hash Tables and requires extra efforts.
2. Doing order statistics, finding closest lower and greater elements, doing range queries are easy to do with BSTs. Like sorting, these operations are not a natural operation with Hash Tables.
3. BSTs are easy to implement compared to hashing, we can easily implement our own customized BST. To implement Hashing, we generally rely on libraries provided by programming languages.
4. With Self-Balancing BSTs, all operations are guaranteed to work in O(Logn) time. But with Hashing, $\Theta(1)$ is average time and some particular operations may be costly, especially when table resizing happens.

This article is contributed by **Himanshu Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/advantages-of-bst-over-hash-table/

Chapter 2

All elements in an array are Same or not?

All elements in an array are Same or not? - GeeksforGeeks

Given an array,
check whether all elements in an array are same or not

Examples:

```
Input : "Geeks", "for", "Geeks"
Output : Not all Elements are Same

Input : 1, 1, 1, 1, 1
Output : All Elements are Same
```

Method 1 (Hashing) We create an empty HashSet, insert all elements into it, then we finally see if size of the HashSet is one or not.

```
 // Java program to check if all array elements are
// same or not.
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class SameElements {
public static boolean areSame(Integer arr[])
    {
        // Put all array elements in a HashSet
        Set<Integer> s = new HashSet<>(Arrays.asList(arr));

        // If all elements are same, size of
```

```
        // HashSet should be 1. As HashSet contains only distinct values.
        return (s.size() == 1);
    }

    // Driver code
public static void main(String[] args)
    {
        Integer[] arr = { 1, 2, 3, 2 };
        if (areSame(arr))
            System.out.println("All Elements are Same");
        else
            System.out.println("Not all Elements are Same");
    }
}
```

Output:

```
Not all Elements are Same
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Method 2 (Compare with first) The idea is simple, we compare every other element with first. If all match with first, we return true. Else we return false.

```
 // Java program to check if all array elements are
// same or not.
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class SameElements {
public static boolean areSame(Integer arr[])
    {
       Integer first = arr[0];
       for (int i=1; i<arr.length; i++)
           if (arr[i] != first)
                return false;
       return true;
    }

    // Driver code
public static void main(String[] args)
    {
        Integer[] arr = { 1, 2, 3, 2 };
        if (areSame(arr))
            System.out.println("All Elements are Same");
```

```
        else
            System.out.println("Not all Elements are Same");
    }
}
```

Output:

```
Not all Elements are Same
```

Time Complexity : O(n)
Auxiliary Space : O(1)

Source

https://www.geeksforgeeks.org/all-elements-in-an-array-are-same-or-not/

Chapter 3

All unique triplets that sum up to a given value

All unique triplets that sum up to a given value - GeeksforGeeks

Given an array and a sum value, find all possible unique triplets in that array whose sum is equal to the given sum value. If no such triplets can be formed from the array, then print "No triplets can be formed", else print all the unique triplets. For example, if the given array is {12, 3, 6, 1, 6, 9} and given sum is 24, then the unique triplets are (3, 9, 12) and (6, 6, 12) whose sum is 24.

Examples:

```
Input : array = {12, 3, 6, 1, 6, 9} sum = 24
Output : [[3, 9, 12], [6, 6, 12]]

Input : array = {-2, 0, 1, 1, 2} sum = 0
Output : [[-2, 0, 2], [-2, 1, 1]]

Input : array = {-2, 0, 1, 1, 2} sum = 10
Output : No triplets can be formed
```

In a previous post, Find a triplet that sum to a given value we have discussed whether the triplets can be formed from the array or not.

Here we need to print all unique set of triplets that sum up to a given value

1. Sort the input array.
2. Find three indexes from the array i, j and k where A[i]+A[j]+A[k] = given sum value.
3. Fix the first element as A[i] and iterate i from 0 to array size – 2.
4. For each iteration of i, take j to be index of the first element in the remaining elements and k to be index of the last element.
5. Check for the triplet combination A[i]+A[j]+A[k] = given sum value.

6. If triplet is obtained (ie., A[i]+A[j]+A[k] = given sum value)
……6.1. Add all the triplet in a TreeSet with ":" separated value to get the unique triplets.
……6.2. increment the second value index
……6.3. decrement the third value index.
……6.4. repeat step 4 & 5 till j < k
7. If A[i]+A[j]+A[k] given sum value, decrement the third value index

C++

```
 // C++ program to find unique triplets
// that sum up to a given value.
#include <bits/stdc++.h>
using namespace std;

// Structure to define a triplet.
struct triplet{
    int first, second, third;
};

// Function to find unique triplets that
// sum up to a given value.
int findTriplets(int nums[], int n, int sum)
{
    int i, j, k;

    // Vector to store all unique triplets.
    vector <triplet> triplets;

    // Set to store already found triplets
    // to avoid duplication.
    unordered_set <string> uniqTriplets;

    // Variable used to hold triplet
    // converted to string form.
    string temp;

    // Variable used to store current
    // triplet which is stored in vector
    // if it is unique.
    triplet newTriplet;

    // Sort the input array.
    sort(nums, nums + n);

    // Iterate over the array from the
    // start and consider it as the
    // first element.
    for(i = 0; i < n - 2; i++){
```

```
// index of the first element in
// the remaining elements.
j = i + 1;

// index of the last element.
k = n - 1;

while(j < k){

    // If sum of triplet is equal to
    // given value, then check if
    // this triplet is unique or not.
    // To check uniqueness, convert
    // triplet to string form and
    // then check if this string is
    // present in set or not. If
    // triplet is unique, then store
    // it in vector.
    if(nums[i] + nums[j] + nums[k] == sum)
    {
        temp = to_string(nums[i]) + " : "
             + to_string(nums[j]) + " : "
                     + to_string(nums[k]);
        if(uniqTriplets.find(temp) ==
                        uniqTriplets.end())
        {
            uniqTriplets.insert(temp);
            newTriplet.first = nums[i];
            newTriplet.second = nums[j];
            newTriplet.third = nums[k];
            triplets.push_back(newTriplet);
        }

        // Increment the first index
        // and decrement the last
        // index of remaining elements.
        j++;
        k--;
    }

    // If sum is greater than given
    // value then to reduce sum
    // decrement the last index.
    else if(nums[i] + nums[j] +
                        nums[k] > sum)
        k--;

    // If sum is less than given value
```

```
            // then to increase sum increment
            // the first index of remaining
            // elements.
            else
                j++;
        }
    }

    // If no unique triplet is found, then
    // return 0.
    if(triplets.size() == 0)
        return 0;

    // Print all unique triplets stored in
    // vector.
    for(i = 0; i < triplets.size(); i++)
    {
        cout << "[" << triplets[i].first
            << ", " << triplets[i].second
          << ", " << triplets[i].third <<"], ";
    }
}

// Driver Function.
int main()
{
    int nums[] = { 12, 3, 6, 1, 6, 9 };
    int n = sizeof(nums) / sizeof(nums[0]);
    int sum = 24;
    if(!findTriplets(nums, n, sum))
        cout << "No triplets can be formed.";

    return 0;
}

// This code is contributed by NIKHIL JINDAL.
```

Java

```
 // Java program to find all triplets with given sum
import java.util.*;

public class triplets {

    // returns all triplets whose sum is equal to sum value
    public static List<List<Integer> > findTriplets(int[] nums, int sum)
    {
```

```
    /* Sort the elements */
    Arrays.sort(nums);

    List<List<Integer> > pair = new ArrayList<>();
    TreeSet<String> set = new TreeSet<String>();
    List<Integer> triplets = new ArrayList<>();

    /* Iterate over the array from the start and
       consider it as the first element*/
    for (int i = 0; i < nums.length - 2; i++) {

        // index of the first element in the
        // remaining elements
        int j = i + 1;

        // index of the last element
        int k = nums.length - 1;

        while (j < k) {

            if (nums[i] + nums[j] + nums[k] == sum) {

                String str = nums[i] + ":" + nums[j] + ":" + nums[k];

                if (!set.contains(str)) {

                    // To check for the unique triplet
                    triplets.add(nums[i]);
                    triplets.add(nums[j]);
                    triplets.add(nums[k]);
                    pair.add(triplets);
                    triplets = new ArrayList<>();
                    set.add(str);
                }

                j++; // increment the second value index
                k--; // decrement the third value index

            } else if (nums[i] + nums[j] + nums[k] < sum)
                j++;

            else // nums[i] + nums[j] + nums[k] > sum
                k--;
        }
    }
    return pair;
}
```

```
    public static void main(String[] args)
    {
        int[] nums = { 12, 3, 6, 1, 6, 9 };
        int sum = 24;

        List<List<Integer> > triplets = findTriplets(nums, sum);

        if (!triplets.isEmpty()) {
            System.out.println(triplets);
        } else {
            System.out.println("No triplets can be formed");
        }
    }
}
```

Output:

```
[[3, 9, 12], [6, 6, 12]]
```

Time Complexity: $O(n^2)$

Improved By : nik1996

Source

https://www.geeksforgeeks.org/unique-triplets-sum-given-value/

Chapter 4

Applications of Hashing

Applications of Hashing - GeeksforGeeks

In this article we will be discussing of applications of hashing.

Hashing provides constant time search, insert and delete operations on average. This is why hashing is one of the most used data structure, example problems are, distinct elements, counting frequencies of items, finding duplicates, etc.

There are many other applications of hashing, including modern day cryptography hash functions. Some of these applications are listed below:

- Message Digest
- Password Verification
- Data Structures(Programming Languages)
- Compiler Operation
- Rabin-Karp Algotithm
- Linking File name and path together

Message Digest:
This is an application of cryptographic Hash Functions. Cryptographic hash functions are the functions which produce an output from which reaching the input is close to impossible. This property of hash functions is called **irreversibility**.

Lets take an **Example**:
Suppose you have to store your files on any of the cloud services available. You have to be sure that the files that you store are not tampered by any third party. You do it by computing "hash" of that file using a Cryptographic hash algorithm. One of the common cryptographic hash algorithms is **SHA 256**. The hash thus computed has a maximum size of 32 bytes. So a computing the
hash of large number of files will not be a problem. You save these hashes on your local machine.

Now, when you download the files, you compute the hash again. Then you match it with the previous hash computed. Therefore, you know whether your files were tampered or not.

If anybody tamper with the file, the hash value of the file will definitely change. Tampering the file without changing the hash is nearly impossible.

Password Verification
Cryptographic hash functions are very commonly used in password verification. Let's understand this using an **Example**:
When you use any online website which requires a user login, you enter your E-mail and password to authenticate that the account you are trying to use belongs to you. When the password is entered, a hash of the password is computed which is then sent to the server for verification of the password. The passwords stored on the server are actually computed hash values of the original passwords. This is done to ensure that when the password is sent from client to server, no sniffing is there.

Data Structures(Programming Languages):
Various programming languages have hash table based Data Structures. The basic idea is to create a key-value pair where key is supposed to be a unique value, whereas value can be same for different keys. This implementation is seen in unordered_set & unordered_map in C++, HashSet & HashMap in java, dict in python etc.

Compiler Operation:
The keywords of a programming language are processed differently than other identifiers. To differentiate between the keywords of a programming language(if, else, for, return etc.) and other identifiers and to successfully compile the program, the compiler stores all these keywords in a set which is implemented using a hash table.

Rabin-Karp Algorithm:
One of the most famous applications of hashing is the Rabin-Karp algorithm. This is basically a string-searching algorithm which uses hashing to find any one set of patterns in a string. A practical application of this algorithm is detecting plagiarism. To know more about Rabin-Karp algo go through Searching for Patterns | Set 3 (Rabin-Karp Algorithm).

Linking File name and path together:
When moving through files on our local system, we observe two very crucial components of a file i.e. file_name and file_path. In order to store the correspondence between file_name and file_path the system uses a map(file_name, file_path)which is implemented using a hash table.

Related articles:

Hashing vs BST
Hashing vs Trie

Source

https://www.geeksforgeeks.org/applications-of-hashing/

Chapter 5

Array elements that appear more than once

Array elements that appear more than once - GeeksforGeeks

Given an integer array, print all repeating elements (Elements that appear more than once) in array. The output should contain elements according to their first occurrences.

Examples:

```
Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45}
Output: 10 45

Input: arr[] = {1, 2, 3, 4, 2, 5}
Output: 2

Input: arr[] = {1, 1, 1, 1, 1}
Output: 1
```

The idea is to **use Hashing** to solve this in O(n) time on average. We store elements and their counts in a hash table. After storing counts, we traverse input array again and print those elements whose counts are more than once. To make sure that every output element is printed only once, we set count as 0 after printing the element.

C++

```
 // C++ program to print all repeating elements
#include <bits/stdc++.h>
using namespace std;

void printRepeating(int arr[], int n)
```

```
{
    // Store elements and their counts in
    // hash table
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Since we want elements in same order,
    // we traverse array again and print
    // those elements that appear more than
    // once.
    for (int i = 0; i < n; i++)
    {
      if (mp[arr[i]] > 1)
      {
          cout << arr[i] << " ";

          // This is tricky, this is done
          // to make sure that the current
          // element is not printed again
          mp[arr[i]] = 0;
      }
    }
}

// Driver code
int main()
{
    int arr[] = { 12, 10, 9, 45, 2, 10, 10, 45 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printRepeating(arr, n);
    return 0;
}
```

Python3

```
 # Python3 program to print
# all repeating elements
def printRepeating(arr,n):

    # Store elements and
    # their counts in
    # hash table
    mp = [0] * 100
    for i in range(0, n):
        mp[arr[i]] += 1

    # Since we want elements
```

```
    # in same order, we
    # traverse array again
    # and print those elements
    # that appear more than once.
    for i in range(0, n):
        if (mp[arr[i]] > 1):
            print(arr[i], end = " ")

            # This is tricky, this
            # is done to make sure
            # that the current element
            # is not printed again
            mp[arr[i]] = 0

# Driver code
arr = [12, 10, 9, 45,
       2, 10, 10, 45]
n = len(arr)
printRepeating(arr, n)

# This code is contributed
# by smita
```

Output :

```
10 45
```

Time Complexity : O(n) under the assumption that hash insert and search functions work in O(1) time.

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/array-elements-that-appear-more-than-once/

Chapter 6

C++ program for hashing with chaining

C++ program for hashing with chaining - GeeksforGeeks

In hashing there is a hash function that maps keys to some values. But these hashing function may lead to collision that is two or more keys are mapped to same value. **Chain hashing** avoids collision. The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

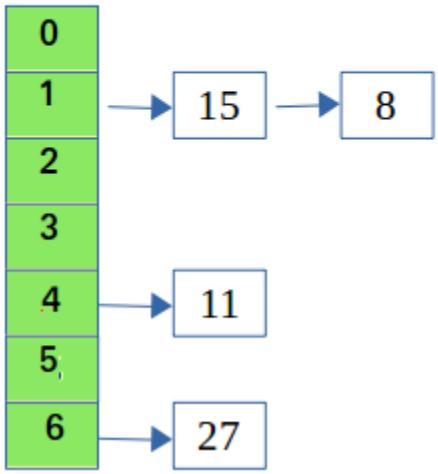
Let's create a hash function, such that our hash table has 'N' number of buckets.
To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.
Example: hashIndex = key % noOfBuckets

Insert: Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list.

Delete: To delete a node from hash table, calculate the hash index for the key, move to the bucket corresponds to the calculated hash index, search the list in the current bucket to find and remove the node with the given key (if found).

Let's say hash table with 7 buckets (0, 1, 2, 3, 4, 5, 6)

Keys arrive in the Order (15, 11 , 27 , 8)



Please refer **Hashing | Set 2 (Separate Chaining)** for details.

We use a list in C++ which is internally implemented as linked list (Faster insertion and deletion).

```
 // CPP program to implement hashing with chaining
#include<iostream>
#include <list>
using namespace std;

class Hash
{
    int BUCKET;    // No. of buckets

    // Pointer to an array containing buckets
    list<int> *table;
public:
    Hash(int V);  // Constructor

    // inserts a key into hash table
    void insertItem(int x);

    // deletes a key from hash table
    void deleteItem(int key);

    // hash function to map values to key
```

```
    int hashFunction(int x) {
        return (x % BUCKET);
    }

    void displayHash();
};

Hash::Hash(int b)
{
    this->BUCKET = b;
    table = new list<int>[BUCKET];
}

void Hash::insertItem(int key)
{
    int index = hashFunction(key);
    table[index].push_back(key);
}

void Hash::deleteItem(int key)
{
  // get the hash index of key
  int index = hashFunction(key);

  // find the key in (inex)th list
  list <int> :: iterator i;
  for (i = table[index].begin();
           i != table[index].end(); i++) {
    if (*i == key)
      break;
  }

  // if key is found in hash table, remove it
  if (i != table[index].end())
    table[index].erase(i);
}

// function to display hash table
void Hash::displayHash() {
  for (int i = 0; i < BUCKET; i++) {
    cout << i;
    for (auto x : table[i])
      cout << " --> " << x;
    cout << endl;
  }
}

// Driver program
```

```
int main()
{
  // array that contains keys to be mapped
  int a[] = {15, 11, 27, 8, 12};
  int n = sizeof(a)/sizeof(a[0]);

  // insert the keys into the hash table
  Hash h(7);   // 7 is count of buckets in
               // hash table
  for (int i = 0; i < n; i++)
    h.insertItem(a[i]);

  // delete 12 from hash table
  h.deleteItem(12);

  // display the Hash table
  h.displayHash();

  return 0;
}
```

Output:

```
0
1 --> 15 --> 8
2
3
4 --> 11
5
6 --> 27
```

Source

https://www.geeksforgeeks.org/c-program-hashing-chaining/

Chapter 7

Change string to a new character set

Change string to a new character set - GeeksforGeeks

Given a 26 letter character set, which is equivalent to character set of English alphabet i.e. (abcd….xyz) and act as a relation. We are also given several sentences and we have to translate them with the help of given new character set.

Examples:

```
New character set : qwertyuiopasdfghjklzxcvbnm
Input : "utta"
Output : geek

Input : "egrt"
Output : code
```

Idea behind conversion of new character set is to use hashing. Perform hashing of new character set where element of set is index and its position will be new alphabet value.

Approach1:

Given New character set = “qwertyuiopasdfghjklzxcvbnm”

1. First character is q, during hashing we will place ‘a’ (for position) at *index q* i.e. (17^{th}).
2. After hashing our new character set is “kvmcnophqrszyijadlegwbuft”.
3. For input “egrt” =
   hash[e -‘a’] = c
   hash[g -‘a’] = o

hash[r -'a'] = d
hash[t -'a'] = e

For "egrt" is equivalent to "code".

C++

```
 // CPP program to change the sentence
// with virtual dictionary
#include<bits/stdc++.h>
using namespace std;

// Converts str to given character set
void conversion(char charSet[], string &str)
{
    int n = str.length();

    // hashing for new character set
    char hashChar[26];
    for (int i = 0; i < 27; i++)
        hashChar[charSet[i]-'a'] = 'a' + i;

    // conversion of new character set
    for (int i = 0; i < n; i++)
        str[i] = hashChar[str[i]-'a'];
}

// Driver code
int main()
{
    char charSet[27] = "qwertyuiopasdfghjklzxcvbnm";
    string str = "egrt";
    conversion(charSet, str);
    cout << str;
    return 0;
}
```

Output:

```
code
```

Approach2:

1.Initialize two strings, one with actual set of alphabets and another with modified one.
2.Get the string to be converted from the user.
3.Retrive the first element of the string, find its index in the modified set of alphabets(eg:0 for 'q').
4.Find the element of same index in the actual set of alphabets and concatenate it with

the result string.
5.Repeat the above steps for all the remaining elements of the input string.
6.Return the result string.

Python3

```
 # Python3 program to change the sentence
#  with virtual dictionary

#function for converting the string
def conversion(charSet,str1):
    s2=""
    for i in str1:
        # find the index of each element of the
        # string in the modified set of alphabets
        # replace the element with the one having the
        # same index in the actual set of alphabets
        s2 += alphabets[charSet.index(i)]

    return s2

# Driver Code
if __name__=='__main__':
    alphabets = "abcdefghijklmnopqrstuvwxyz"
    charSet= "qwertyuiopasdfghjklzxcvbnm"
    str1 = "egrt"
    print(conversion(charSet,str1))

#This code is contributed by PradeepEswar
```

Output:

```
code
```

Improved By : PradeepEswar

Source

https://www.geeksforgeeks.org/change-string-to-a-new-character-set/

Chapter 8

Change the array into a permutation of numbers from 1 to n

Change the array into a permutation of numbers from 1 to n - GeeksforGeeks

Given an array A of n elements. We need to change the array into a permutation of numbers from 1 to n using minimum replacements in the array.
Examples:

```
Input : A[] = {2, 2, 3, 3}
Output : 2 1 3 4
Explanation:
To make it a permutation of 1 to 4, 1 and 4 are
missing from the array. So replace 2, 3 with
1 and 4.

Input :  A[] = {1, 3, 2}
Output : 1 3 2

Input : A[] = {10, 1, 2}
Output : 3 1 2
```

Approach: Observe that we don't need to change the numbers which are in the range [1, n] and which are distinct(has only one occurrence). So, we use a greedy approach. If we meet the number we have never met before and this number is between 1 and n, we leave this number unchanged. And remove the duplicate elements and add the missing elements in the range [1, n]. Also replace the numbers, not in the range.

C++

```
 // CPP program to make a permutation of numbers
// from 1 to n using minimum changes.
#include <bits/stdc++.h>
using namespace std;

void makePermutation(int a[], int n)
{
    // Store counts of all elements.
    unordered_map<int, int> count;
    for (int i = 0; i < n; i++)
        count[a[i]]++;

    int next_missing = 1;
    for (int i = 0; i < n; i++) {
        if (count[a[i]] != 1 || a[i] > n || a[i] < 1) {
            count[a[i]]--;

            // Find next missing element to put
            // in place of current element.
            while (count.find(next_missing) != count.end())
                next_missing++;

            // Replace with next missing and insert the
            // missing element in hash.
            a[i] = next_missing;
            count[next_missing] = 1;
        }
    }
}

// Driver Code
int main()
{
    int A[] = { 2, 2, 3, 3 };
    int n = sizeof(A) / sizeof(A[0]);
    makePermutation(A, n);
    for (int i = 0; i < n; i++)
        cout << A[i] << " ";
    return 0;
}
```

Python3

```
 # Python3 code to make a permutation
# of numbers from 1 to n using
```

```
# minimum changes.

def makePermutation (a, n):

    # Store counts of all elements.
    count = dict()
    for i in range(n):
        if count.get(a[i]):
            count[a[i]] += 1
        else:
            count[a[i]] = 1;

    next_missing = 1
    for i in range(n):
        if count[a[i]] != 1 or a[i] > n or a[i] < 1:
            count[a[i]] -= 1

            # Find next missing element to put
            # in place of current element.
            while count.get(next_missing):
                next_missing+=1

            # Replace with next missing and
            # insert the missing element in hash.
            a[i] = next_missing
            count[next_missing] = 1

# Driver Code
A = [ 2, 2, 3, 3 ]
n = len(A)
makePermutation(A, n)

for i in range(n):
    print(A[i], end = " ")

# This code is contributed by "Sharad_Bhardwaj".
```

Output:

```
1 2 4 3
```

Source

https://www.geeksforgeeks.org/change-array-permutation-numbers-1-n/

Chapter 9

Character replacement after removing duplicates from a string

Character replacement after removing duplicates from a string - GeeksforGeeks

Given a string. The task is to replace each character of the minimized string by a character present at index '*IND*' of the original string. The minimized string is the string obtained by removing all duplicates from the original string keeping the order of elements same.

IND for any index in the minimized string is calculated as:

> **IND = (square of ascii value of minimized string character) % (length of original string)**

Examples:

```
Input : geeks
Output : sesg
Explanation :   minimized string = geks
                length of original string = 5
                ascii value of g = 103
                IND for g = (103*103) % 5 = 4
                replacement character for g = s
                character 's' present at index 4 of original string
Similarly,
replacement character for e = e
replacement character for k = s
replacement character for s = g
```

```
Input : helloworld
Output : oeoeeoh
```

Approach:

Below is the step by step algorithm for string minimization:

```
1. Initalize flagchar[26] = {0}
2. for i=0 to str.length()-1
3.     ch = str[i]
4.     if flagchar[ch-97] == 0 then
5.         mstr = mstr + ch
6.         flagchar[ch-97] = 1
7.     End if
8. End of loop
9. return mstr  // minimized string
```

Algorithm for character replacement:

```
1. Replace each character of minimized string as described
   in the problem statement and example
2. Compute final string
```

Below is the implementation of above approach:

```
 // C++ program for character replacement
// after string minimization
#include <bits/stdc++.h>
using namespace std;

// Function to minimize string
string minimize(string str)
{
    string mstr = " ";
    int l, i, flagchar[26] = { 0 };
    char ch;

    l = str.length();

    // duplicate characters are removed
    for (i = 0; i < str.length(); i++)
    {
        ch = str.at(i);

        // checks if character has previously occurred or not
```

```
        // if not then add it to the minimized string 'mstr'
        if (flagchar[ch-97] == 0)
        {
            mstr = mstr + ch;
            flagchar[ch-97] = 1;
        }
    }

    return mstr; // minimized string
}

// Utility function to print the
// minimized, replaced string
void replaceMinimizeUtil(string str)
{
    string minimizedStr, finalStr = "";
    int i, index, l;
    char ch;
    l = str.length();

    minimizedStr = minimize(str); // minimized string

    // Creating final string by replacing character
    for (i = 0; i < minimizedStr.length(); i++)
    {
        ch = minimizedStr.at(i);

        // index calculation
        index = (ch*ch) % l;

        finalStr = finalStr + str.at(index);
    }

    cout << "Final string: " << finalStr; // final string
}

// Driver program
int main()
{
    string str = "geeks";

    replaceMinimizeUtil(str);

    return 0;
}
```

Output:

```
Final string: ssesg
```

Time Complexity: O(n)

Source

https://www.geeksforgeeks.org/character-replacement-string-minimization/

Chapter 10

Check for Palindrome after every character replacement Query

Check for Palindrome after every character replacement Query - GeeksforGeeks

Given a string **str** and **Q** queries. Each query contains a pair of integers (i1, i2) and a character 'ch'. We need to replace characters at indexes i1 and i2 with new character 'ch' and then tell if string str is palindrome or not. (0 <= i1, i2 < string_length) Examples:

```
Input : str = "geeks"  Q = 2
        query 1: i1 = 3 ,i2 = 0, ch = 'e'
        query 2: i1 = 0 ,i2 = 2 , ch = 's'
Output : query 1: "NO"
         query 2: "YES"
Explanation :
        In query 1 : i1 = 3 , i2 = 0 ch = 'e'
                     After replacing char at index i1, i2
                     str[3] = 'e', str[0] = 'e'
                     string become "eeees" which is not
                     palindrome so output "NO"
        In query 2 : i1 = 0 i2 = 2  ch = 's'
                     After replacing char at index i1 , i2
                      str[0] = 's', str[2] = 's'
                     string become "seses" which is
                     palindrome so output "YES"

Input : str = "jasonamat"  Q = 3
        query 1: i1 = 3, i2 = 8 ch = 'j'
        query 2: i1 = 2, i2 = 6 ch = 'n'
```

```
        query 3: i1 = 3, i2 = 7 ch = ‘a’
Output :
       query 1: "NO"
       query 2: "NO"
       query 3: "YES"
```

A **Simple solution** is that for each query , we replace character at indexes (i1 & i2) with a new character ‘ch’ and then check if string is palindrome or not.

Below is C++ implementation of above idea

```
 // C++ program to find if string becomes palindrome
// after every query.
#include<bits/stdc++.h>
using namespace std;

// Function to check if string is Palindrome or Not
bool IsPalindrome(string &str)
{
    int n = strlen(str);
    for (int i = 0; i < n/2 ; i++)
        if (str[i] != str[n-1-i])
            return false;
    return true;
}

// Takes two inputs for Q queries. For every query, it
// prints Yes if string becomes palindrome and No if not.
void Query(string &str, int Q)
{
    int i1, i2;
    char ch;

    // Process all queries one by one
    for (int q = 1 ; q <= Q ; q++ )
    {
        cin >> i1 >> i2 >> ch;

        // query 1: i1 = 3 ,i2 = 0, ch = 'e'
        // query 2: i1 = 0 ,i2 = 2 , ch = 's'
        // replace character at index i1 & i2 with new 'ch'
        str[i1] = str[i2] = ch;

        // check string is palindrome or not
        (isPalindrome(str)== true) ? cout << "YES" << endl :
                                     cout << "NO" << endl;
    }
```

```
}

// Driver program
int main()
{
    char str[] = "geeks";
    int Q = 2;
    Query(str, Q);
    return 0;
}
```

Input:

```
3 0 e
0 2 s
```

Output:

```
"NO"
"YES"
```

Time complexity O(Q*n) (n is length of string)

An **efficient solution** is to use hashing. We create an empty hash set that stores indexes that are unequal in palindrome (**Note**: " we have to store indexes only first half of string that are unequal ").

```
Given string "str" and length 'n'.
Create an empty set S and store unequal indexes in first half.
Do following for each query :
   1. First replace character at indexes i1 & i2 with
      new char "ch"

   2. If i1 and/or i2 are/is greater than n/2 then convert
      into first half index(es)

   3. In this step we make sure that S contains maintains
      unequal indexes of first half.
      a) If str[i1] == str [n - 1 - i1] means i1 becomes
         equal after replacement, remove it from S (if present)
         Else add i1 to S
      b) Repeat step a) for i2 (replace i1 with i2)

   4. If S is empty then string is palindrome else NOT
```

Below is C++ implementation of above idea

```
 // C++/c program check if given string is palindrome
// or not after every query
#include<bits/stdc++.h>
using namespace std;

// This function makes sure that set S contains
// unequal characters from first half. This is called
// for every character.
void addRemoveUnequal(string &str, int index, int n,
                              unordered_set<int> &S)
{
    // If character becomes equal after query
    if (str[index] == str[n-1-index])
    {
        // Remove the current index from set if it
        // is present
        auto it = S.find(index);
        if (it != S.end())
            S.erase(it) ;
    }

    // If not equal after query, insert it into set
    else
        S.insert(index);
}

// Takes two inputs for Q queries. For every query, it
// prints Yes if string becomes palindrome and No if not.
void Query(string &str, int Q)
{
    int n = str.length();

    // create an empty set that store indexes of
    // unequal location in palindrome
    unordered_set<int> S;

    // we store indexes that are unequal in palindrome
    // traverse only first half of string
    for (int i=0; i<n/2; i++)
        if (str[i] != str[n-1-i])
            S.insert(i);

    // traversal the query
    for (int q=1; q<=Q; q++)
    {
        // query 1: i1 = 3, i2 = 0, ch = 'e'
```

```
        // query 2: i1 = 0, i2 = 2, ch = 's'
        int i1, i2;
        char ch;
        cin >> i1 >> i2 >> ch;

        // Replace characters at indexes i1 & i2 with
        // new char 'ch'
        str[i1] = str [i2] = ch;

        // If i1 and/or i2 greater than n/2
        // then convert into first half index
        if (i1 > n/2)
            i1 = n- 1 -i1;
        if (i2 > n/2)
            i2 = n -1 - i2;

        // call addRemoveUnequal function to insert and remove
        // unequal indexes
        addRemoveUnequal(str, i1 , n, S );
        addRemoveUnequal(str, i2 , n, S );

        // if set is not empty then string is not palindrome
        S.empty()? cout << "YES\n" : cout << "NO\n";
    }
}

// Driver program
int main()
{
    string str = "geeks";
    int Q = 2 ;
    Query(str, Q);
    return 0;
}
```

Input:

```
3 0 e
0 2 s
```

Output:

```
"NO"
"YES"
```

Time Complexity : O(Q + n) under the assumption that set insert, delete and find operations take O(1) time.

Source

https://www.geeksforgeeks.org/check-for-palindrome-after-every-character-replacement-query/

Chapter 11

Check if a given array contains duplicate elements within k distance from each other

Check if a given array contains duplicate elements within k distance from each other - GeeksforGeeks

Given an unsorted array that may contain duplicates. Also given a number k which is smaller than size of array. Write a function that returns true if array contains duplicates within k distance.

Examples:

```
Input: k = 3, arr[] = {1, 2, 3, 4, 1, 2, 3, 4}
Output: false
All duplicates are more than k distance away.

Input: k = 3, arr[] = {1, 2, 3, 1, 4, 5}
Output: true
1 is repeated at distance 3.

Input: k = 3, arr[] = {1, 2, 3, 4, 5}
Output: false

Input: k = 3, arr[] = {1, 2, 3, 4, 4}
Output: true
```

A **Simple Solution** is to run two loops. The outer loop picks every element 'arr[i]' as a starting element, the inner loop compares all elements which are within k distance of 'arr[i]'. The time complexity of this solution is O(kn).

We can solve this problem **in Θ(n) time using Hashing.** The idea is to one by add elements to hash. We also remove elements which are at more than k distance from current element. Following is detailed algorithm.

1) Create an empty hashtable.
2) Traverse all elements from left from right. Let the current element be 'arr[i]'
….a) If current element 'arr[i]' is present in hashtable, then return true.
….b) Else add arr[i] to hash and remove arr[i-k] from hash if i is greater than or equal to k

C/C++

```
 #include<bits/stdc++.h>
using namespace std;

/* C++ program to Check if a given array contains duplicate
   elements within k distance from each other */
bool checkDuplicatesWithinK(int arr[], int n, int k)
{
    // Creates an empty hashset
    set<int> myset;

    // Traverse the input array
    for (int i = 0; i < n; i++)
    {
        // If already present n hash, then we found
        // a duplicate within k distance
        if (myset.find(arr[i]) != myset.end())
            return true;

        // Add this item to hashset
        myset.insert(arr[i]);

        // Remove the k+1 distant item
        if (i >= k)
            myset.erase(arr[i-k]);
    }
    return false;
}

// Driver method to test above method
int main ()
{
    int arr[] = {10, 5, 3, 4, 3, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    if (checkDuplicatesWithinK(arr, n, 3))
        cout << "Yes";
    else
        cout << "No";
}
```

```
//This article is contributed by Chhavi
```

JAVA

```
 /* Java program to Check if a given array contains duplicate
   elements within k distance from each other */
import java.util.*;

class Main
{
    static boolean checkDuplicatesWithinK(int arr[], int k)
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array
        for (int i=0; i<arr.length; i++)
        {
            // If already present n hash, then we found
            // a duplicate within k distance
            if (set.contains(arr[i]))
               return true;

            // Add this item to hashset
            set.add(arr[i]);

            // Remove the k+1 distant item
            if (i >= k)
              set.remove(arr[i-k]);
        }
        return false;
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        if (checkDuplicatesWithinK(arr, 3))
           System.out.println("Yes");
        else
           System.out.println("No");
    }
}
```

Output:

```
Yes
```

This article is contributed by **Anuj**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/check-given-array-contains-duplicate-elements-within-k-distance/

Chapter 12

Check if a string is Isogram or not

Check if a string is Isogram or not - GeeksforGeeks

Given a word or phrase, check if it is isogram or not. An Isogram is a word in which no letter occurs more than once.

Examples:

```
Input : Machine
Output : True
"Machine" does not have any character repeating,
it is an Isogram

Input : Geek
Output : False
"Geek" has 'e' as repeating character,
it is not an Isogram
```

C++

```
 // C++ program to check
// if a given string is isogram or not
#include <bits/stdc++.h>
using namespace std;

// Function to check
// if a given string is isogram or not
string is_isogram(string str)
{
    int len = str.length();
```

```
    // Convert the string in lower case letters
    for (int i = 0; i < len; i++)
        str[i] = tolower(str[i]);

    sort(str.begin(), str.end());

    for (int i = 0; i < len; i++) {
        if (str[i] == str[i + 1])
            return "False";
    }
    return "True";
}

// driver program
int main()
{
    string str1 = "Machine";
    cout << is_isogram(str1) << endl;

    string str2 = "isogram";
    cout << is_isogram(str2) << endl;

    string str3 = "GeeksforGeeks";
    cout << is_isogram(str3) << endl;

    string str4 = "Alphabet";
    cout << is_isogram(str4) << endl;

    return 0;
}

// Contributed by nuclode
```

Java

```
 // Java program to check
// if a given string is isogram or not
import java.io.*;
import java.util.*;

class GFG {
    // Function to check
    // if a given string is isogram or not
    static boolean is_isogram(String str)
    {
        // Convert the string in lower case letters
        str = str.toLowerCase();
```

```
        int len = str.length();

        char arr[] = str.toCharArray();

        Arrays.sort(arr);
        for (int i = 0; i < len - 1; i++) {
            if (arr[i] == arr[i + 1])
                return false;
        }
        return true;
    }

    // driver program
    public static void main(String[] args)
    {
        String str1 = "Machine";
        System.out.println(is_isogram(str1));

        String str2 = "isogram";
        System.out.println(is_isogram(str2));

        String str3 = "GeeksforGeeks";
        System.out.println(is_isogram(str3));

        String str4 = "Alphabet";
        System.out.println(is_isogram(str4));
    }
}

// Contributed by Pramod Kumar
```

Python

```
 # Python program to check
# if a word is isogram or not
def is_isogram(word):

    # Convert the word or sentence in lower case letters.
    clean_word = word.lower()

    # Make an empty list to append unique letters
    letter_list = []

    for letter in clean_word:

        # If letter is an alphabet then only check
        if letter.isalpha():
            if letter in letter_list:
```

```
                return False
            letter_list.append(letter)

    return True

if __name__ == '__main__':
    print(is_isogram("Machine"))
    print(is_isogram("isogram"))
    print(is_isogram("GeeksforGeeks"))
    print(is_isogram("Alphabet "))
```

C#

```
 // C# program to check if a given
// string is isogram or not
using System;

public class GFG {

    // Function to check if a given
    // string is isogram or not
    static bool is_isogram(string str)
    {
        // Convert the string in lower case letters
        str = str.ToLower();
        int len = str.Length;

        char[] arr = str.ToCharArray();

        Array.Sort(arr);
        for (int i = 0; i < len - 1; i++) {
            if (arr[i] == arr[i + 1])
                return false;
        }
        return true;
    }

    // driver program
    public static void Main()
    {
        string str1 = "Machine";
        Console.WriteLine(is_isogram(str1));

        string str2 = "isogram";
        Console.WriteLine(is_isogram(str2));

        string str3 = "GeeksforGeeks";
        Console.WriteLine(is_isogram(str3));
```

```
        string str4 = "Alphabet";
        Console.WriteLine(is_isogram(str4));
    }
}

// This code is contributed by Sam007
```

Output:

```
True
True
False
False
```

Another approach : In this, count of characters of string are stored in hashmap, and wherever it is found to be greater than 1 for any char, return false else return true.

```
 // CPP code to check string is isogram or not
#include <bits/stdc++.h>

using namespace std;

// function to check isogram
bool check_isogram(string str)
{

    int length = str.length();
    int mapHash[26] = { 0 };

    // loop to store count of chars and check if it is greater than 1
    for (int i = 0; i < length; i++) {
        mapHash[str[i] - 'a']++;

        // if count > 1, return false
        if (mapHash[i] > 1) {
            return false;
        }
    }

    return true;
}

// Driver code
int main()
{
```

```
    string str = "geeks";
    string str2 = "computer";

    // checking str as isogram
    if (check_isogram(str)) {
        cout << "True" << endl;
    }
    else {
        cout << "False" << endl;
    }

    // checking str2 as isogram
    if (check_isogram(str2)) {
        cout << "True" << endl;
    }
    else {
        cout << "False" << endl;
    }

    return 0;
}
```

Output :

```
False
True
```

// Thanks **Sahil Bansal** for suggesting the above method.

Improved By : Sam007

Source

https://www.geeksforgeeks.org/check-string-isogram-not/

Chapter 13

Check if all array elements are distinct

Check if all array elements are distinct - GeeksforGeeks

Given an array, check whether all elements in an array are distinct or not.

Examples:

```
Input : 1, 3, 2, 4
Output : Yes

Input : "Geeks", "for", "Geeks"
Output : No

Input : "All", "Not", "Equal"
Output : Yes
```

One **simple solution** is to use two nested loops. For every element, check if it repeats or not. If any element repeats, return false. If no element repeats, return false.

An **efficient solution** is to Hashing. We put all array elements in a HashSet. If size of HashSet remains same as array size, then we return true.

C++

```
 // C++ program to check if all array elements are distinct
#include<bits/stdc++.h>

using namespace std;

bool areDistinct(vector<int> arr)
    {
```

```
        int n = arr.size();
         // Put all array elements in a map
         map<int, bool> s ;
         for(int i=0; i<n; i++){
           s.insert(pair<int, int>(arr[i],true));
         }

         // If all elements are distinct, size of
         // map should be same array.
         return (s.size() == arr.size());
    }
 // Driver code
int main(){

 std::vector<int>arr = {1, 2, 3, 2};

 if(areDistinct(arr)){
    cout<<"All Elements are Distinct";
 }
 else{
    cout<<"Not all Elements are Distinct";
 }

return 0;
}
```

Java

```
 // Java program to check if all array elements are
// distinct or not.
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class DistinctElements {
    public static boolean areDistinct(Integer arr[])
    {
         // Put all array elements in a HashSet
         Set s = new HashSet(Arrays.asList(arr));

         // If all elements are distinct, size of
         // HashSet should be same array.
         return (s.size() == arr.length);
    }

    // Driver code
    public static void main(String[] args)
    {
```

```
        Integer[] arr = {1, 2, 3, 2};
        if (areDistinct(arr))
            System.out.println("All Elements are Distinct");
        else
            System.out.println("Not all Elements are Distinct");
    }
}
```

Output:

```
Not all Elements are Distinct
```

Improved By : Subhashni Singh

Source

https://www.geeksforgeeks.org/check-if-all-array-elements-are-distinct/

Chapter 14

Check if an array can be divided into pairs whose sum is divisible by k

Check if an array can be divided into pairs whose sum is divisible by k - GeeksforGeeks

Given an array of integers and a number k, write a function that returns true if given array can be divided into pairs such that sum of every pair is divisible by k.

Examples:

```
Input: arr[] = {9, 7, 5, 3},
           k = 6
Output: True
We can divide array into (9, 3) and
(7, 5). Sum of both of these pairs
is a multiple of 6.

Input: arr[] = {92, 75, 65, 48, 45, 35},
           k = 10
Output: True
We can divide array into (92, 48), (75, 65)
and (45, 35). Sum of all these pairs is a
multiple of 10.

Input: arr[] = {91, 74, 66, 48}, k = 10
Output: False
```

A **Simple Solution** is to iterate through every element arr[i]. Find if there is another not yet visited element that has remainder as (k – arr[i]%k). If there is no such element,

return false. If a pair is found, then mark both elements as visited. Time complexity of this solution is O(n^2 and it requires O(n) extra space.

An **Efficient Solution** is to use Hashing.

```
1) If length of given array is odd, return false.
    An odd length array cannot be divided in pairs.
2) Traverse input array and count occurrences of
    all remainders.
      freq[arr[i] % k]++
3) Traverse input array again.
   a) Find remainder of current element.
   b) If remainder divides k into two halves, then
      there must be even occurrences of it as it
      forms  pair with itself only.
   c) If remainder is 0, then then there must be
      even occurrences.
   c) Else, number of occurrences of current
      remainder must be equal to number of
       occurrences of "k - current remainder".
```

Time complexity of above algorithm is O(n).

Below implementation uses map in C++ STL. The map is typically implemented using Red-Black Tree and takes O(Log n) time for access. Therefore time complexity of below implementation is O(n Log n), but the algorithm can be easily implemented in O(n) time using hash table.

C++

```
 // A C++ program to check if arr[0..n-1] can be divided
// in pairs such that every pair is divisible by k.
#include <bits/stdc++.h>
using namespace std;

// Returns true if arr[0..n-1] can be divided into pairs
// with sum divisible by k.
bool canPairs(int arr[], int n, int k)
{
    // An odd length array cannot be divided into pairs
    if (n & 1)
        return false;

    // Create a frequency array to count occurrences
    // of all remainders when divided by k.
    map<int, int> freq;

    // Count occurrences of all remainders
```

```
    for (int i = 0; i < n; i++)
        freq[arr[i] % k]++;

    // Traverse input array and use freq[] to decide
    // if given array can be divided in pairs
    for (int i = 0; i < n; i++)
    {
        // Remainder of current element
        int rem = arr[i] % k;

        // If remainder with current element divides
        // k into two halves.
        if  (2*rem == k)
        {
            // Then there must be even occurrences of
            // such remainder
            if (freq[rem] % 2 != 0)
                return false;
        }

        // If remainder is 0, then there must be two
        // elements with 0 remainder
        else if (rem == 0)
        {
           if (freq[rem] & 1)
               return false;
        }

        // Else number of occurrences of remainder
        // must be equal to number of occurrences of
        // k - remainder
        else if (freq[rem] != freq[k - rem])
            return false;
    }
    return true;
}

/* Driver program to test above function */
int main()
{
    int arr[] =  {92, 75, 65, 48, 45, 35};
    int k = 10;
    int n = sizeof(arr)/sizeof(arr[0]);
    canPairs(arr, n, k)? cout << "True": cout << "False";
    return 0;
}
```

Java

```
import java.util.HashMap;

public class Divisiblepair
{
    // Returns true if arr[0..n-1] can be divided into pairs
    // with sum divisible by k.
    static boolean canPairs(int ar[], int k)
    {
        // An odd length array cannot be divided into pairs
        if (ar.length % 2 == 1)
            return false;

        // Create a frequency array to count occurrences
        // of all remainders when divided by k.
        HashMap<Integer, Integer> hm = new HashMap<>();

        // Count occurrences of all remainders
        for (int i = 0; i < ar.length; i++)
        {
            int rem = ar[i] % k;
            if (!hm.containsKey(rem))
            {
                hm.put(rem, 0);
            }
            hm.put(rem, hm.get(rem) + 1);
        }

        // Traverse input array and use freq[] to decide
        // if given array can be divided in pairs
        for (int i = 0; i < ar.length; i++)
        {
             // Remainder of current element
            int rem = ar[i] % k;

            // If remainder with current element divides
            // k into two halves.
            if (2 * rem == k)
            {
                // Then there must be even occurrences of
                // such remainder
                if (hm.get(rem) % 2 == 1)
                    return false;
            }

            // If remainder is 0, then there must be two
            // elements with 0 remainder
            else if (rem == 0)
```

```
            {
                // Then there must be even occurrences of
                // such remainder
                if (hm.get(rem) % 2 == 1)
                    return false;
            }

            // Else number of occurrences of remainder
            // must be equal to number of occurrences of
            // k - remainder
            else
            {
                if (hm.get(k - rem) != hm.get(rem))
                    return false;
            }
        }
        return true;
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        int arr[] = { 92, 75, 65, 48, 45, 35 };
        int k = 10;
        boolean ans = canPairs(arr, k);
        if (ans)
            System.out.println("True");
        else
            System.out.println("False");

    }
}

// This code is contributed by Rishabh Mahrsee
```

Output:

```
True
```

This article is contributed by **Priyanka**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/check-if-an-array-can-be-divided-into-pairs-whose-sum-is-divisible-by-k/

Chapter 15

Check if any anagram of a string is palindrome or not

Check if any anagram of a string is palindrome or not - GeeksforGeeks

We have given a anagram string and we have to check whether it can be made palindrome o not.

Examples:

```
Input : geeksforgeeks
Output : No
There is no palindrome anagram of
given string

Input  : geeksgeeks
Output : Yes
There are palindrome anagrams of
given string. For example kgeeseegk
```

This problem is basically same as Check if characters of a given string can be rearranged to form a palindrome. We can do it in O(n) time using a count array. Following are detailed steps.
1) Create a count array of alphabet size which is typically 256. Initialize all values of count array as 0.
2) Traverse the given string and increment count of every character.
3) Traverse the count array and if the count array has more than one odd values, return false. Otherwise return true.

C++

```
 #include <iostream>
using namespace std;
#define NO_OF_CHARS 256

/* function to check whether characters of a string
   can form a palindrome */
bool canFormPalindrome(string str)
{
    // Create a count array and initialize all
    // values as 0
    int count[NO_OF_CHARS] = { 0 };

    // For each character in input strings,
    // increment count in the corresponding
    // count array
    for (int i = 0; str[i]; i++)
        count[str[i]]++;

    // Count odd occurring characters
    int odd = 0;
    for (int i = 0; i < NO_OF_CHARS; i++) {
        if (count[i] & 1)
            odd++;

        if (odd > 1)
            return false;
    }

    // Return true if odd count is 0 or 1,
    return true;
}

/* Driver program to test to print printDups*/
int main()
{
    canFormPalindrome("geeksforgeeks") ? cout << "Yes\n" : cout << "No\n";
    canFormPalindrome("geeksogeeks") ? cout << "Yes\n" : cout << "No\n";
    return 0;
}
```

Java

```
 // Java program to Check if any anagram
// of a string is palindrome or not
public class GFG {
    static final int NO_OF_CHARS = 256;

    /* function to check whether characters of
```

```
      a string can form a palindrome */
    static boolean canFormPalindrome(String str)
    {
        // Create a count array and initialize
        // all values as 0
        int[] count = new int[NO_OF_CHARS];

        // For each character in input strings,
        // increment count in the corresponding
        // count array
        for (int i = 0; i < str.length(); i++)
            count[str.charAt(i)]++;

        // Count odd occurring characters
        int odd = 0;
        for (int i = 0; i < NO_OF_CHARS; i++) {
            if ((count[i] & 1) != 0)
                odd++;

            if (odd > 1)
                return false;
        }

        // Return true if odd count is 0 or 1,
        return true;
    }

    /* Driver program to test to print printDups*/
    public static void main(String args[])
    {
        System.out.println(canFormPalindrome("geeksforgeeks")
                                ? "Yes"
                                : "No");
        System.out.println(canFormPalindrome("geeksogeeks")
                                ? "Yes"
                                : "No");
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 NO_OF_CHARS = 256

""" function to check whether characters of a string
   can form a palindrome """
def canFormPalindrome(string):
```

```
    # Create a count array and initialize all
    # values as 0
    count = [0 for i in range(NO_OF_CHARS)]

    # For each character in input strings,
    # increment count in the corresponding
    # count array
    for i in string:
        count[ord(i)] += 1

    # Count odd occurring characters
    odd = 0
    for i in range(NO_OF_CHARS):
        if (count[i] & 1):
            odd += 1

        if (odd > 1):
            return False

    # Return true if odd count is 0 or 1,
    return True

# Driver program to test to print printDups
if(canFormPalindrome("geeksforgeeks")):
    print "Yes"
else:
    print "No"
if(canFormPalindrome("geeksogeeks")):
    print "Yes"
else:
    print "NO"

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to Check if any anagram
// of a string is palindrome or not
using System;

public class GFG {

    static int NO_OF_CHARS = 256;

    /* function to check whether
    characters of a string can form
    a palindrome */
    static bool canFormPalindrome(string str)
```

```
    {

        // Create a count array and
        // initialize all values as 0
        int[] count = new int[NO_OF_CHARS];

        // For each character in input
        // strings, increment count in
        // the corresponding count array
        for (int i = 0; i < str.Length; i++)
            count[str[i]]++;

        // Count odd occurring characters
        int odd = 0;
        for (int i = 0; i < NO_OF_CHARS; i++) {
            if ((count[i] & 1) != 0)
                odd++;

            if (odd > 1)
                return false;
        }

        // Return true if odd count
        // is 0 or 1,
        return true;
    }

    // Driver program
    public static void Main()
    {
        Console.WriteLine(
            canFormPalindrome("geeksforgeeks")
                              ? "Yes" : "No");

        Console.WriteLine(
            canFormPalindrome("geeksogeeks")
                              ? "Yes" : "No");
    }
}

// This code is contributed by vt_m.
```

Output:

```
No
Yes
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/check-anagram-string-palindrome-not/

Chapter 16

Check if array contains contiguous integers with duplicates allowed

Check if array contains contiguous integers with duplicates allowed - GeeksforGeeks

Given an array of **n** integers(duplicates allowed). Print "Yes" if it is a set of contiguous integers else print "No".

Examples:

```
Input : arr[] = {5, 2, 3, 6, 4, 4, 6, 6}
Output : Yes
The elements form a contiguous set of integers
which is {2, 3, 4, 5, 6}.

Input : arr[] = {10, 14, 10, 12, 12, 13, 15}
Output : No
```

Source: Amazon interview Experience | Set 416.

We have discussed different solutions for distinct elements in below post.
Check if array elements are consecutive

A **simple solution** is to first sort the array. Then traverse the array to check if all consecutive elements differ at most by one.

C

```
 // Sorting based C++ implementation
// to check whether the array
```

```
// contains a set of contiguous
// integers
#include <bits/stdc++.h>
using namespace std;

// function to check whether
// the array contains a set
// of contiguous integers
bool areElementsContiguous(int arr[], int n)
{
    // Sort the array
    sort(arr, arr+n);

     // After sorting, check if
     // current element is either
     // same as previous or is
     // one more.
    for (int i = 1; i < n; i++)
        if (arr[i] - arr[i-1] > 1)
            return false;

    return true;
}

// Driver program to test above
int main()
{
    int arr[] = { 5, 2, 3, 6,
                    4, 4, 6, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    if (areElementsContiguous(arr, n))
        cout << "Yes";

    else
        cout << "No";

    return 0;
}
```

Java

```
 // Sorting based Java implementation
// to check whether the array
// contains a set of contiguous
// integers
import java.util.*;
```

```
class GFG {

    // function to check whether
    // the array contains a set
    // of contiguous integers
    static boolean areElementsContiguous(int arr[],
                                         int n)
    {
       // Sort the array
       Arrays.sort(arr);

       // After sorting, check if
       // current element is either
       // same as previous or is
       // one more.
       for (int i = 1; i < n; i++)
         if (arr[i] - arr[i-1] > 1)
              return false;

       return true;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = { 5, 2, 3, 6,
                      4, 4, 6, 6 };
        int n = arr.length;

        if (areElementsContiguous(arr, n))
            System.out.println("Yes");

        else
            System.out.println("No");

    }

}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Sorting based Python implementation
# to check whether the array
# contains a set of contiguous integers

def areElementsContiguous(arr, n):
```

```
    # Sort the array
    arr.sort()

    # After sorting, check if
    # current element is either
    # same as previous or is
    # one more.
    for i in range(1,n):
        if (arr[i] - arr[i-1] > 1) :
            return 0
    return 1

# Driver code
arr = [ 5, 2, 3, 6, 4, 4, 6, 6 ]
n = len(arr)
if areElementsContiguous(arr, n): print("Yes")
else: print("No")

# This code is contributed by 'Ansu Kumari'.
```

C#

```
 // Sorting based C# implementation
// to check whether the array
// contains a set of contiguous
// integers
using System;

class GFG {

    // function to check whether
    // the array contains a set
    // of contiguous integers
    static bool areElementsContiguous(int []arr,
                                      int n)
    {
    // Sort the array
    Array.Sort(arr);

    // After sorting, check if
    // current element is either
    // same as previous or is
    // one more.
    for (int i = 1; i < n; i++)
        if (arr[i] - arr[i - 1] > 1)
            return false;

    return true;
```

```
    }

    // Driver program
    public static void Main()
    {
        int []arr = { 5, 2, 3, 6,
                      4, 4, 6, 6 };
        int n = arr.Length;

        if (areElementsContiguous(arr, n))
            Console.WriteLine("Yes");

        else
            Console.WriteLine("No");

    }

}

// This code is contributed by Vt_m.
```

Output:

```
Yes
```

Time Complexity : O(n Log n)

Efficient solution using visited array
1) Find minimum and maximum elements.
2) Create a visited array of size max – min + 1. Initialize this array as false.
3) Traverse the given array and mark visited[arr[i] – min] as true for every element arr[i].
4) Traverse visited array and return true if all values are true. Else return false.

C++

```
 // C++ implementation to
// check whether the array
// contains a set of
// contiguous integers
#include <bits/stdc++.h>
using namespace std;

// function to check
// whether the array
// contains a set of
// contiguous integers
bool areElementsContiguous(int arr[], int n)
```

```
{
    // Find maximum and
    // minimum elements.
    int max = *max_element(arr, arr + n);
    int min = *min_element(arr, arr + n);

    int m = max - min + 1;

    // There should be at least
    // m elements in array to
    // make them contiguous.
    if (m > n)
        return false;

    // Create a visited array
    // and initialize false.
    bool visited[m];
    memset(visited, false, sizeof(visited));

    // Mark elements as true.
    for (int i=0; i<n; i++)
    visited[arr[i] - min] = true;

    // If any element is not
    // marked, all elements
    // are not contiguous.
    for (int i=0; i<m; i++)
    if (visited[i] == false)
            return false;

    return true;
}

// Driver program
int main()
{
    int arr[] = { 5, 2, 3, 6,
                  4, 4, 6, 6 };

    int n = sizeof(arr) / sizeof(arr[0]);

    if (areElementsContiguous(arr, n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to
// check whether the array
// contains a set of
// contiguous integers
import java.util.*;

class GFG {

    // function to check
    // whether the array
    // contains a set of
    // contiguous integers
    static boolean areElementsContiguous(int arr[],
                                         int n)
    {
        // Find maximum and
        // minimum elements.
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;

        for(int i = 0; i < n; i++)
        {
            max = Math.max(max, arr[i]);
            min = Math.min(min, arr[i]);
        }

        int m = max - min + 1;

        // There should be at least
        // m elements in aaray to
        // make them contiguous.
        if (m > n)
            return false;

        // Create a visited array
        // and initialize false.
        boolean  visited[] = new boolean[n];


        // Mark elements as true.
        for (int i = 0; i < n; i++)
           visited[arr[i] - min] = true;

        // If any element is not
        // marked, all elements
        // are not contiguous.
```

```
        for (int i = 0; i < m; i++)
           if (visited[i] == false)
                return false;

        return true;
    }

    /* Driver program  */
    public static void main(String[] args)
    {
        int arr[] = { 5, 2, 3, 6,
                      4, 4, 6, 6 };

        int n = arr.length;

        if (areElementsContiguous(arr, n))
            System.out.println("Yes");

        else
            System.out.println("No");

      }
}
```

Python3

```
 # Python3 implementation to
# check whether the array
# contains a set of
# contiguous integers

# function to check
# whether the array
# contains a set of
# contiguous integers
def areElementsContiguous(arr, n):

    # Find maximum and
    # minimum elements.
    max1 = max(arr)
    min1 = min(arr)

    m = max1 - min1 + 1

    # There should be at least
    # m elements in array to
    # make them contiguous.
    if (m > n):
```

```
        return False

    # Create a visited array
    # and initialize fals

    visited = [0] * m

    # Mark elements as true.
    for i in range(0,n) :
        visited[arr[i] - min1] = True

    # If any element is not
    # marked, all elements
    # are not contiguous.
    for i in range(0, m):
        if (visited[i] == False):
            return False

    return True

# Driver program
arr = [5, 2, 3, 6, 4, 4, 6, 6 ]
n = len(arr)

if (areElementsContiguous(arr, n)):
    print("Yes")
else:
    print("No")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# implementation to check whether
// the array contains a set of
// contiguous integers
using System;

class GFG {

    // function to check whether the
    // array contains a set of
    // contiguous integers
    static bool areElementsContiguous(
                        int []arr, int n)
    {

        // Find maximum and
```

```
    // minimum elements.
    int max = int.MinValue;
    int min = int.MaxValue;

    for(int i = 0; i < n; i++)
    {
        max = Math.Max(max, arr[i]);
        min = Math.Min(min, arr[i]);
    }

    int m = max - min + 1;

    // There should be at least
    // m elements in aaray to
    // make them contiguous.
    if (m > n)
        return false;

    // Create a visited array
    // and initialize false.
    bool []visited = new bool[n];


    // Mark elements as true.
    for (int i = 0; i < n; i++)
        visited[arr[i] - min] = true;

    // If any element is not
    // marked, all elements
    // are not contiguous.
    for (int i = 0; i < m; i++)
        if (visited[i] == false)
            return false;

    return true;
}

/* Driver program */
public static void Main()
{
    int []arr = { 5, 2, 3, 6,
                4, 4, 6, 6 };

    int n = arr.Length;

    if (areElementsContiguous(arr, n))
        Console.Write("Yes");
    else
```

```
            Console.Write("No");
    }
}

// This code is contributed by nitin mittal.
```

Output:

```
Yes
```

Time Complexity : O(n)

Efficient solution using hash table
Insert all the elements in the hash table. Now pick the first element and keep on incrementing in its value by 1 till you find a value not present in the hash table. Again pick the first element and keep on decrementing in its value by 1 till you find a value not present in the hash table. Get the **count** of elements (obtained by this process) which are present in the hash table. If the count equals hash size print "Yes" else "No".

C++

```
 // C++ implementation to check whether the array
// contains a set of contiguous integers
#include <bits/stdc++.h>
using namespace std;

// Function to check whether the array contains
// a set of contiguous integers
bool areElementsContiguous(int arr[], int n)
{
    // Storing elements of 'arr[]' in a hash
    // table 'us'
    unordered_set<int> us;
    for (int i = 0; i < n; i++)
        us.insert(arr[i]);

    // as arr[0] is present in 'us'
    int count = 1;

    // starting with previous smaller element
    // of arr[0]
    int curr_ele = arr[0] - 1;

    // if 'curr_ele' is present in 'us'
    while (us.find(curr_ele) != us.end()) {

        // increment count
```

```
        count++;

        // update 'curr_ele"
        curr_ele--;
    }

    // starting with next greater element
    // of arr[0]
    curr_ele = arr[0] + 1;

    // if 'curr_ele' is present in 'us'
    while (us.find(curr_ele) != us.end()) {

        // increment count
        count++;

        // update 'curr_ele"
        curr_ele++;
    }

    // returns true if array contains a set of
    // contiguous integers else returns false
    return (count == (int)(us.size()));
}

// Driver program to test above
int main()
{
    int arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (areElementsContiguous(arr, n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Java implementation to check whether the array
// contains a set of contiguous integers
import java.io.*;
import java.util.*;

class GFG {
    // Function to check whether the array
    // contains a set of contiguous integers
```

```
static Boolean areElementsContiguous(int arr[], int n)
{
    // Storing elements of 'arr[]' in
    // a hash table 'us'
    HashSet<Integer> us = new HashSet<Integer>();

    for (int i = 0; i < n; i++)
        us.add(arr[i]);

    // As arr[0] is present in 'us'
    int count = 1;

    // Starting with previous smaller
    // element of arr[0]
    int curr_ele = arr[0] - 1;

    // If 'curr_ele' is present in 'us'
    while (us.contains(curr_ele) == true) {

        // increment count
        count++;

        // update 'curr_ele"
        curr_ele--;
    }

    // Starting with next greater
    // element of arr[0]
    curr_ele = arr[0] + 1;

    // If 'curr_ele' is present in 'us'
    while (us.contains(curr_ele) == true) {

        // increment count
        count++;

        // update 'curr_ele"
        curr_ele++;
    }

    // Returns true if array contains a set of
    // contiguous integers else returns false
    return (count == (us.size()));
}

// Driver Code
public static void main(String[] args)
{
```

```
        int arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };
        int n = arr.length;

        if (areElementsContiguous(arr, n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by 'Gitanjali'.
```

Python

```
 # Python implementation to check whether the array
# contains a set of contiguous integers

# Function to check whether the array
# contains a set of contiguous integers
def areElementsContiguous(arr):
    # Storing elements of 'arr[]' in a hash table 'us'
    us = set()
    for i in arr: us.add(i)

    # As arr[0] is present in 'us'
    count = 1

    # Starting with previous smaller element of arr[0]
    curr_ele = arr[0] - 1

    # If 'curr_ele' is present in 'us'
    while curr_ele in us:

        # Increment count
        count += 1

        # Update 'curr_ele"
        curr_ele -= 1

    # Starting with next greater element of arr[0]
    curr_ele = arr[0] + 1

    # If 'curr_ele' is present in 'us'
    while curr_ele in us:

        # Increment count
        count += 1
```

```
        # Update 'curr_ele"
        curr_ele += 1

    # Returns true if array contains a set of
    # contiguous integers else returns false
    return (count == len(us))

# Driver code
arr = [ 5, 2, 3, 6, 4, 4, 6, 6 ]
if areElementsContiguous(arr): print("Yes")
else: print("No")

# This code is contributed by 'Ansu Kumari'
```

Output :

```
Yes
```

Time Complexity: O(n).
Auxiliary Space: O(n).

This method requires only one traversal of given array. It traverses hash table after array traversal (hash table contains only distinct elements).

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/check-array-contains-contiguous-integers-duplicates-allowed/

Chapter 17

Check if characters of a given string can be rearranged to form a palindrome

Check if characters of a given string can be rearranged to form a palindrome - GeeksforGeeks

Given a string, Check if characters of the given string can be rearranged to form a palindrome.
For example characters of “geeksogeeks” can be rearranged to form a palindrome “geeksoskeeg”, but characters of “geeksforgeeks” cannot be rearranged to form a palindrome.

A set of characters can form a palindrome if at most one character occurs odd number of times and all characters occur even number of times.

A simple solution is to run two loops, the outer loop picks all characters one by one, the inner loop counts number of occurrences of the picked character. We keep track of odd counts. Time complexity of this solution is $O(n^2)$.

We can do it in O(n) time using a count array. Following are detailed steps.
1) Create a count array of alphabet size which is typically 256. Initialize all values of count array as 0.
2) Traverse the given string and increment count of every character.
3) Traverse the count array and if the count array has more than one odd values, return false. Otherwise return true.

Below is the implementation of above approach.

C++

```
 // C++ implementation to check if
// characters of a given string can
// be rearranged to form a palindrome
#include<bits/stdc++.h>
```

```
using namespace std;
#define NO_OF_CHARS 256

/* function to check whether characters of a string can form
   a palindrome */
bool canFormPalindrome(string str)
{
    // Create a count array and initialize all
    // values as 0
    int count[NO_OF_CHARS] = {0};

    // For each character in input strings,
    // increment count in the corresponding
    // count array
    for (int i = 0; str[i];  i++)
        count[str[i]]++;

    // Count odd occurring characters
    int odd = 0;
    for (int i = 0; i < NO_OF_CHARS; i++)
    {
        if (count[i] & 1)
            odd++;

        if (odd > 1)
            return false;
    }

    // Return true if odd count is 0 or 1,
    return true;
}

/* Driver program*/
int main()
{
  canFormPalindrome("geeksforgeeks")? cout << "Yes\n":
                                      cout << "No\n";
  canFormPalindrome("geeksogeeks")? cout << "Yes\n":
                                    cout << "No\n";
  return 0;
}
```

Java

```
 // Java implementation to check if
// characters of a given string can
// be rearranged to form a palindrome
import java.io.*;
```

```
import java.util.*;
import java.math.*;

class GFG {

static int NO_OF_CHARS = 256;

    /* function to check whether characters
    of a string can form a palindrome */
    static boolean canFormPalindrome(String str) {

    // Create a count array and initialize all
    // values as 0
    int count[] = new int[NO_OF_CHARS];
    Arrays.fill(count, 0);

    // For each character in input strings,
    // increment count in the corresponding
    // count array
    for (int i = 0; i < str.length(); i++)
    count[(int)(str.charAt(i))]++;

    // Count odd occurring characters
    int odd = 0;
    for (int i = 0; i < NO_OF_CHARS; i++)
    {
    if ((count[i] & 1) == 1)
        odd++;

    if (odd > 1)
        return false;
    }

    // Return true if odd count is 0 or 1,
    return true;
}

// Driver program
public static void main(String args[])
{
    if (canFormPalindrome("geeksforgeeks"))
    System.out.println("Yes");
    else
    System.out.println("No");

    if (canFormPalindrome("geeksogeeks"))
    System.out.println("Yes");
    else
```

```
    System.out.println("No");
}
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python3 implementation to check if
# characters of a given string can
# be rearranged to form a palindrome

NO_OF_CHARS = 256

# function to check whether characters
# of a string can form a palindrome
def canFormPalindrome(st) :

    # Create a count array and initialize
    # all values as 0
    count = [0] * (NO_OF_CHARS)

    # For each character in input strings,
    # increment count in the corresponding
    # count array
    for i in range( 0, len(st)) :
        count[ord(st[i])] = count[ord(st[i])] + 1

    # Count odd occurring characters
    odd = 0

    for i in range(0, NO_OF_CHARS ) :
        if (count[i] & 1) :
            odd = odd + 1

        if (odd > 1) :
            return False

    # Return true if odd count is 0 or 1,
    return True

# Driver program
if(canFormPalindrome("geeksforgeeks")) :
    print("Yes")
else :
    print("No")

if(canFormPalindrome("geeksogeeks")) :
```

```
    print("Yes")
else :
    print("No")

# This code is contributed by Nikita Tiwari.
```

Output:

```
No
Yes
```

This article is contributed by Abhishek. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/check-characters-given-string-can-rearranged-form-palindrome/

Chapter 18

Check if frequency of all characters can become same by one removal

Check if frequency of all characters can become same by one removal - GeeksforGeeks

Given a string which contains lower alphabetic characters, we need to remove at most one character from this string in such a way that frequency of each distinct character becomes same in the string.
Examples:

```
Input  : str = "xyyz"
Output : Yes
We can remove character 'y' from above
string to make the frequency of each
character same.

Input : str = "xyyzz"
Output : Yes
We can remove character 'x' from above
string to make the frequency of each
character same.

Input : str = "xxxxyyzz"
Output : No
It is not possible to make frequency of
each character same just by removing at
most one character from above string.
```

Main thing to observe in this problem is that position of characters does not matter here so we will count the frequency of characters, if all of them are same then we are done and

there is no need to remove any character to make frequency of characters same Otherwise we can iterate over all characters one by one and decrease their frequency by one, if all frequencies become same then we will flag that it is possible to make character frequency same by at most one removal and if frequencies doesn't match then we will increase that frequency again and loop for other characters.

C++

```
 //    C++ program to get same frequency character
// string by removal of at most one char
#include <bits/stdc++.h>
using namespace std;
#define M 26

// Utility method to get index of character ch
// in lower alphabet characters
int getIdx(char ch)
{
    return (ch - 'a');
}

// Returns true if all non-zero elements
// values are same
bool allSame(int freq[], int N)
{
    int same;

    //    get first non-zero element
    int i;
    for (i = 0; i < N; i++)
    {
        if (freq[i] > 0)
        {
            same = freq[i];
            break;
        }
    }

    //    check equality of each element with variable same
    for (int j = i+1; j < N; j++)
        if (freq[j] > 0 && freq[j] != same)
            return false;

    return true;
}

// Returns true if we can make all character
// frequencies same
```

```
bool possibleSameCharFreqByOneRemoval(string str)
{
    int l = str.length();

    //    fill frequency array
    int freq[M] = {0};
    for (int i = 0; i < l; i++)
        freq[getIdx(str[i])]++;

    //    if all frequencies are same, then return true
    if (allSame(freq, M))
        return true;

    /*  Try decreasing frequency of all character
        by one and then    check all equality of all
        non-zero frequencies */
    for (char c = 'a'; c <= 'z'; c++)
    {
        int i = getIdx(c);

        // Check character only if it occurs in str
        if (freq[i] > 0)
        {
            freq[i]--;

            if (allSame(freq, M))
                return true;
            freq[i]++;
        }
    }

    return false;
}

// Driver code to test above methods
int main()
{
    string str = "xyyzz";
    if (possibleSameCharFreqByOneRemoval(str))
        cout << "Yes";
    else
        cout << "No";
}
```

Java

```
 // Java program to get same frequency character
// string by removal of at most one char
```

```
public class GFG {

    static final int M = 26;

    // Utility method to get index of character ch
    // in lower alphabet characters
    static int getIdx(char ch)
    {
        return (ch - 'a');
    }

    // Returns true if all non-zero elements
    // values are same
    static boolean allSame(int freq[], int N)
    {
        int same = 0;

        //  get first non-zero element
        int i;
        for (i = 0; i < N; i++)
        {
            if (freq[i] > 0)
            {
                same = freq[i];
                break;
            }
        }

        //  check equality of each element with
        // variable same
        for (int j = i+1; j < N; j++)
            if (freq[j] > 0 && freq[j] != same)
            return false;

    return true;
}

    // Returns true if we can make all character
    // frequencies same
    static boolean possibleSameCharFreqByOneRemoval(String str)
    {
        int l = str.length();

        //  fill frequency array
        int[] freq = new int[M];

        for (int i = 0; i < l; i++)
            freq[getIdx(str.charAt(i))]++;
```

```
        //  if all frequencies are same, then return true
        if (allSame(freq, M))
            return true;

        /*  Try decreasing frequency of all character
            by one and then check all equality of all
            non-zero frequencies */
        for (char c = 'a'; c <= 'z'; c++)
        {
            int i = getIdx(c);

            // Check character only if it occurs in str
            if (freq[i] > 0)
            {
                freq[i]--;

                if (allSame(freq, M))
                    return true;
                freq[i]++;
            }
        }

        return false;
    }

    // Driver code to test above methods
    public static void main(String args[])
    {
        String str = "xyyzz";
        if (possibleSameCharFreqByOneRemoval(str))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
// This code is contributed by Sumit Ghosh
```

C#

```
 // C# program to get same frequency
// character string by removal of
// at most one char
using System;

class GFG
{
    static int M = 26;
```

```
    // Utility method to get
    // index of character ch
    // in lower alphabet characters
    static int getIdx(char ch)
    {
        return (ch - 'a');
    }

    // Returns true if all
    // non-zero elements
    // values are same
    static bool allSame(int[] freq,
                        int N)
    {
        int same = 0;

        // get first non-zero element
        int i;
        for (i = 0; i < N; i++)
        {
            if (freq[i] > 0)
            {
                same = freq[i];
                break;
            }
        }

        // check equality of
        // each element with
        // variable same
        for (int j = i + 1; j < N; j++)
            if (freq[j] > 0 &&
                freq[j] != same)
            return false;

    return true;
}

    // Returns true if we
    // can make all character
    // frequencies same
    static bool possibleSameCharFreqByOneRemoval(string str)
    {
        int l = str.Length;

        // fill frequency array
        int[] freq = new int[M];
```

```
        for (int i = 0; i < l; i++)
            freq[getIdx(str[i])]++;

        // if all frequencies are same,
        // then return true
        if (allSame(freq, M))
            return true;

        /* Try decreasing frequency of all
            character by one and then check
            all equality of all non-zero
            frequencies */
        for (char c = 'a'; c <= 'z'; c++)
        {
            int i = getIdx(c);

            // Check character only if
            // it occurs in str
            if (freq[i] > 0)
            {
                freq[i]--;

                if (allSame(freq, M))
                    return true;
                freq[i]++;
            }
        }

        return false;
    }

    // Driver code
    public static void Main()
    {
        string str = "xyyzz";
        if (possibleSameCharFreqByOneRemoval(str))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP program to get same frequency
// character string by removal of at
// most one char
$M = 26;

// Utility method to get index
// of character ch in lower
// alphabet characters
function getIdx($ch)
{
    return ($ch - 'a');
}

// Returns true if all
// non-zero elements
// values are same
function allSame(&$freq, $N)
{
    // get first non-zero element
    for ($i = 0; $i < $N; $i++)
    {
        if ($freq[$i] > 0)
        {
            $same = $freq[$i];
            break;
        }
    }

    // check equality of each
    // element with variable same
    for ($j = $i + 1; $j < $N; $j++)
        if ($freq[$j] > 0 &&
            $freq[$j] != $same)
            return false;

    return true;
}

// Returns true if we
// can make all character
// frequencies same
function possibleSameCharFreqByOneRemoval($str)
{
    global $M;
    $l = strlen($str);

    // fill frequency array
```

```
    $freq = array_fill(0, $M, NULL);
    for ($i = 0; $i < $l; $i++)
        $freq[getIdx($str[$i])]++;

    // if all frequencies are same,
    // then return true
    if (allSame($freq, $M))
        return true;

    /* Try decreasing frequency of all
        character by one and then check
        all equality of all non-zero
        frequencies */
    for ($c = 'a'; $c <= 'z'; $c++)
    {
        $i = getIdx($c);

        // Check character only
        // if it occurs in str
        if ($freq[$i] > 0)
        {
            $freq[$i]--;

            if (allSame($freq, $M))
                return true;
            $freq[$i]++;
        }
    }

    return false;
}

// Driver code
$str = "xyyzz";
if (possibleSameCharFreqByOneRemoval($str))
echo "Yes";
else
echo "No";

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
Yes
```

Time Complexity : O(n) assuming alphabet size is constant.

Improved By : ChitraNayal

Source

https://www.geeksforgeeks.org/check-if-frequency-of-all-characters-can-become-same-by-one-removal/

Chapter 19

Check if the first and last digit of the smallest number forms a prime

Check if the first and last digit of the smallest number forms a prime - GeeksforGeeks

Given an array arr[] containing numbers from 0 to 9 only, the task is to form the minimum possible number from the given digits and then check if the first and last digit of the number thus created can be rearranged to form a prime number or not.

Examples:

> **Input:** arr[]={2, 6, 4, 9}
> **Output:** Minimum number: 2469
> Prime number combinations: 29
> The first and last digits are 2 and 9 respectively. The combinations are 29 and 92. Only 29 is prime.
>
> **Input:** arr[]={2, 6, 4, 3, 1, 7}
> **Output:** Minimum number: 123467
> Prime number combinations: 17 71
> The first and last digits are 1 and 7 respectively. The combinations are 17 and 71, and both are primes

Approach:

1. Create a hash of size 10 to store the number of occurrences of the digits in the given array into the hash table.
2. Print the digits the number of times they occur in descending order starting from the digit 0. It is similar to Smallest number by rearranging digits of a given number.
3. For the prime checking, check if the number formed using the first and last digits is prime or not. Do the same for its reverse.

Below is the implementation of above approach:

C++

```
 // C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// function to check prime
int isPrime(int n)
{
    int i, c = 0;
    for (i = 1; i < n / 2; i++) {
        if (n % i == 0)
            c++;
    }
    if (c == 1)
        return 1;
    else
        return 0;
}

// Function to generate smallest possible
// number with given digits
void findMinNum(int arr[], int n)
{
    // Declare a hash array of size 10
    // and initialize all the elements to zero
    int first = 0, last = 0, num, rev, i;
    int hash[10] = { 0 };

    // store the number of occurrences of the digits
    // in the given array into the hash table
    for (int i = 0; i < n; i++) {
        hash[arr[i]]++;
    }

    // Traverse the hash in ascending order
    // to print the required number
    cout << "Minimum number: ";
    for (int i = 0; i <= 9; i++) {

        // Print the number of times a digits occurs
        for (int j = 0; j < hash[i]; j++)
            cout << i;
    }

    cout << endl;
```

```
    // extracting the first digit
    for (i = 0; i <= 9; i++) {
        if (hash[i] != 0) {
            first = i;
            break;
        }
    }
    // extracting the last digit
    for (i = 9; i >= 0; i--) {
        if (hash[i] != 0) {
            last = i;
            break;
        }
    }

    num = first * 10 + last;
    rev = last * 10 + first;

    // printing the prime combinations
    cout << "Prime combinations: ";
    if (isPrime(num) && isPrime(rev))
        cout << num << " " << rev;

    else if (isPrime(num))
        cout << num;

    else if (isPrime(rev))
        cout << rev;

    else
        cout << "No combinations exist";
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 4, 7, 8};
    findMinNum(arr, 5);

    return 0;
}
```

Java

```
// Java implementation of above approach
```

```
import java.io.*;

class SmallPrime
{

// function to check prime
static boolean isPrime(int n)
{
    int i, c = 0;
    for (i = 1; i < n / 2; i++)
    {
        if (n % i == 0)
            c++;
    }
    if (c == 1)
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Function to generate smallest possible
// number with given digits
static void findMinNum(int arr[], int n)
{
    // Declare a hash array of size 10
    // and initialize all the elements to zero
    int first = 0, last = 0, num, rev, i;
    int hash[] = new int[10];

    // store the number of occurrences of the digits
    // in the given array into the hash table
    for ( i = 0; i < n; i++)
    {
        hash[arr[i]]++;
    }

    // Traverse the hash in ascending order
    // to print the required number
    System.out.print("Minimum number: ");
    for ( i = 0; i <= 9; i++)
    {

        // Print the number of times a digits occurs
        for (int j = 0; j < hash[i]; j++)
```

```
            System.out.print(i);

    }
    System.out.println();

    System.out.println();
    // extracting the first digit
    for (i = 0; i <= 9; i++)
    {
        if (hash[i] != 0)
        {
            first = i;
            break;
        }
    }
    // extracting the last digit
    for (i = 9; i >= 0; i--)
    {
        if (hash[i] != 0)
        {
            last = i;
            break;
        }
    }

    num = first * 10 + last;
    rev = last * 10 + first;

    // printing the prime combinations
    System.out.print( "Prime combinations: ");
    if (isPrime(num) && isPrime(rev))
    {
        System.out.println(num + " " + rev);
    }
    else if (isPrime(num))
    {
        System.out.println(num);
    }
    else if (isPrime(rev))
    {
        System.out.println(rev);
    }

    else
    {
        System.out.println("No combinations exist");
    }
}
```

```
// Driver code

    public static void main (String[] args)
    {
       SmallPrime smallprime = new SmallPrime();
       int arr[] = {1, 2, 4, 7, 8};
       smallprime.findMinNum(arr, 5);
    }
}

// This code has been contributed by inder_verma.
```

Output:

```
Minimum number: 12478
Prime combinations: No combinations exist
```

Improved By : inderDuMCA

Source

https://www.geeksforgeeks.org/check-if-the-first-and-last-digit-of-the-smallest-number-forms-a-prime/

Chapter 20

Check if the given permutation is a valid DFS of graph

Check if the given permutation is a valid DFS of graph - GeeksforGeeks

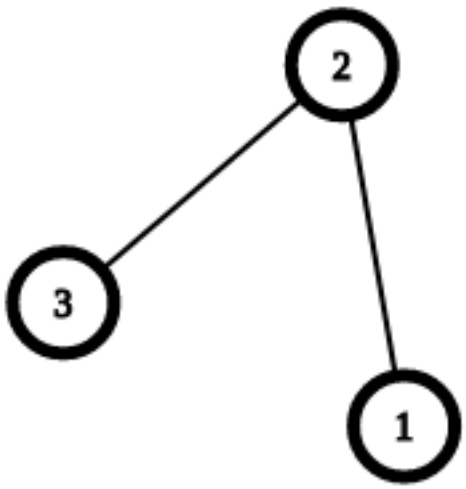
Given a graph with N nodes numbered from 1 to N and M edges and an array of numbers from 1 to N. Check if it is possible to obtain any permutation of array by applying DFS (Depth First Traversal) on given graph.

Prerequisites : DFS | Map in CPP

Examples :

```
Input : N = 3, M = 2
        Edges are:
        1) 1-2
        2) 2-3
        P = {1, 2, 3}
Output : YES
Explanation :
Since there are edges between
1-2 and 2-3, therefore we can
have DFS in the order 1-2-3

Input : N = 3, M = 2
        Edges are:
        1) 1-2
        2) 2-3
        P = {1, 3, 2}
Output : NO
Explanation :
Since there is no edge between 1 and 3,
the DFS traversal is not possible
```

in the order of given permutation.



Possible Permutations in which whole graph can be traversed are:
1) 1 2 3
2) 3 2 1

Approach : We assume that the input graph is represented as adjacency list. The idea is to first sort all adjacency lists according to input order, then traverse the given graph starting from first node in given given permutation. If we visit all vertices in same order, then given permutation is a valid DFS.

1. Store the indexes of each number in the given permutation in a Hash map.
2. Sort every adjacency list according to the indexes of permutation since there is need to maintain the order.
3. Perform the Depth First Traversal Search with source node as 1st number of given permutation.
4. Keep a counter variable and at every recursive call, check if the counter has reached the number of nodes, i.e. N and set the flag as 1. If the flag is 0 after complete DFS, answer is 'NO' otherwise 'YES'

Below is the implementation of above approach :

```
 // CPP program to check if given
// permutation can be obtained
// upon DFS traversal on given graph
#include <bits/stdc++.h>
using namespace std;

// To track of DFS is valid or not.
bool flag = false;
```

```
// HashMap to store the indexes
// of given permutation
map<int, int> mp;

// Comparator function for sort
bool cmp(int a, int b)
{
    // Sort according ascending
    // order of indexes
    return mp[a] < mp[b];
}

// Graph class represents a undirected
// using adjacency list representation
class Graph
{
    int V; // No. of vertices
    int counter; // Counter variable

public:
    // Pointer to an array containing
    // adjacency lists
    list<int>* adj;

    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int u, int v);

    // DFS traversal of the vertices
    // reachable from v
    void DFS(int v, int Perm[]);
};

Graph::Graph(int V)
{
    this->V = V;
    this->counter = 0;
    adj = new list<int>[V + 1];
}

void Graph::addEdge(int u, int v)
{
    adj[u].push_back(v); // Add v to u's list.
    adj[v].push_back(u); // Add u to v's list
}

// DFS traversal of the
```

```
// vertices reachable from v.
void Graph::DFS(int v, int Perm[])
{
    // Increment counter for
    // every node being traversed
    counter++;

    // Check if counter has
    // reached number of vertices
    if (counter == V) {

        // Set flag to 1
        flag = 1;
        return;
    }

    // Recur for all vertices adjacent
    // to this vertices only if it
    // lies in the given permutation
    list<int>::iterator i;
    for (i = adj[v].begin();
         i != adj[v].end(); i++)
    {

        // if the current node equals to
        // current element of permutation
        if (*i == Perm[counter])
            DFS(*i, Perm);
    }
}

// Returns true if P[] is a valid DFS of given
// graph. In other words P[] can be obtained by
// doing a DFS of the graph.
bool checkPermutation(int N, int M,
      vector<pair<int, int> > V, int P[])
{
    // Create the required graph with
    // N vertices and M edges
    Graph G(N);

    // Add Edges to Graph G
    for (int i = 0; i < M; i++)
        G.addEdge(V[i].first, V[i].second);

    for (int i = 0; i < N; i++)
        mp[P[i]] = i;
```

```
    // Sort every adjacency
    // list according to HashMap
    for (int i = 1; i <= N; i++)
        G.adj[i].sort(cmp);

    // Call DFS with source node as P[0]
    G.DFS(P[0], P);

    // If Flag has been set to 1, means
    // given permutation is obtained
    // by DFS on given graph
    return flag;
}

// Driver code
int main()
{
    // Number of vertices and number of edges
    int N = 3, M = 2;

    // Vector of pair to store edges
    vector<pair<int, int> > V;

    V.push_back(make_pair(1, 2));
    V.push_back(make_pair(2, 3));

    int P[] = { 1, 2, 3 };

    // Return the answer
    if (checkPermutation(N, M, V, P))
        cout << "YES" << endl;
    else
        cout << "NO" << endl;

    return 0;
}
```

Output:

```
YES
```

Source

https://www.geeksforgeeks.org/check-given-permutation-valid-dfs-graph/

Chapter 21

Check if two arrays are equal or not

Check if two arrays are equal or not - GeeksforGeeks

Given two given arrays of equal length, the task is to find if given arrays are equal or not. Two arrays are said to be equal if both of them contain same set of elements, arrangements (or permutation) of elements may be different though.

Note : If there are repetitions, then counts of repeated elements must also be same for two array to be equal.

Examples :

```
Input  : arr1[] = {1, 2, 5, 4, 0};
         arr2[] = {2, 4, 5, 0, 1};
Output : Yes

Input  : arr1[] = {1, 2, 5, 4, 0, 2, 1};
         arr2[] = {2, 4, 5, 0, 1, 1, 2};
Output : Yes

Input : arr1[] = {1, 7, 1};
        arr2[] = {7, 7, 1};
Output : No
```

A **simple solution** is to sort both array and then linearly compare elements.

C/C++

```
// C++ program to find given two array
```

```
// are equal or not
#include<bits/stdc++.h>
using namespace std;

// Returns true if arr1[0..n-1] and arr2[0..m-1]
// contain same elements.
bool areEqual(int arr1[], int arr2[], int n, int m)
{
    // If lengths of array are not equal means
    // array are not equal
    if (n != m)
        return false;

    // Sort both arrays
    sort(arr1, arr1+n);
    sort(arr2, arr2+m);

    // Linearly compare elements
    for (int i=0; i<n; i++)
         if (arr1[i] != arr2[i])
            return false;

    // If all elements were same.
    return true;
}

// Driver Code
int main()
{
    int arr1[] = { 3, 5, 2, 5, 2};
    int arr2[] = { 2, 3, 5, 5, 2};
    int n = sizeof(arr1)/sizeof(int);
    int m = sizeof(arr2)/sizeof(int);

    if (areEqual(arr1, arr2, n, m))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java program to find given two array
// are equal or not
import java.io.*;
import java.util.*;
```

```
class GFG
{
    // Returns true if arr1[0..n-1] and arr2[0..m-1]
    // contain same elements.
    public static boolean areEqual(int arr1[], int arr2[])
    {
        int n = arr1.length;
        int m = arr2.length;

        // If lengths of array are not equal means
        // array are not equal
        if (n != m)
            return false;

        // Sort both arrays
        Arrays.sort(arr1);
        Arrays.sort(arr2);

        // Linearly compare elements
        for (int i=0; i<n; i++)
            if (arr1[i] != arr2[i])
                return false;

        // If all elements were same.
        return true;
    }

    //Driver code
    public static void main (String[] args)
    {
        int arr1[] = { 3, 5, 2, 5, 2};
        int arr2[] = { 2, 3, 5, 5, 2};

        if (areEqual(arr1, arr2))
            System.out.println("Yes");
        else
            System.out.println("No");

    }
}
```

C#

```
 // C# program to find given two array
// are equal or not
using System;

class GFG {
```

```
    // Returns true if arr1[0..n-1] and
    // arr2[0..m-1] contain same elements.
    public static bool areEqual(int []arr1,
                                int []arr2)
    {
        int n = arr1.Length;
        int m = arr2.Length;

        // If lengths of array are not
        // equal means array are not equal
        if (n != m)
            return false;

        // Sort both arrays
        Array.Sort(arr1);
        Array.Sort(arr2);

        // Linearly compare elements
        for (int i = 0; i < n; i++)
            if (arr1[i] != arr2[i])
                return false;

        // If all elements were same.
        return true;
    }

    // Driver code
    public static void Main ()
    {
        int []arr1 = { 3, 5, 2, 5, 2};
        int []arr2 = { 2, 3, 5, 5, 2};

        if (areEqual(arr1, arr2))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find given
// two array are equal or not
```

```
// Returns true if arr1[0..n-1]
// and arr2[0..m-1] contain same elements.
function areEqual( $arr1, $arr2, $n, $m)
{
    // If lengths of array
    // are not equal means
    // array are not equal
    if ($n != $m)
        return false;

    // Sort both arrays
    sort($arr1);
    sort($arr2);

    // Linearly compare elements
    for ( $i = 0; $i < $n; $i++)
        if ($arr1[$i] != $arr2[$i])
            return false;

    // If all elements were same.
    return true;
}

// Driver Code
$arr1 = array( 3, 5, 2, 5, 2);
$arr2 = array( 2, 3, 5, 5, 2);
$n = count($arr1);
$m = count($arr2);

if (areEqual($arr1, $arr2, $n, $m))
    echo "Yes";
else
    echo "No";

// This code is contributed by anuj_67.
?>
```

Output :

```
Yes
```

Time Complexity : O(n log n)
Auxiliary Space : O(1)

An **Efficient solution** of this approach is to use hashing. We store all elements of arr1[] and their counts in a hash table. Then we traverse arr2[] and check if count of every element in arr2[] matches with count in arr1[].

Below is C++ implementation of above idea. We use unordered_map to store counts.

C/C++

```
 // C++ program to find given two array
// are equal or not using hashing technique
#include<bits/stdc++.h>
using namespace std;

// Returns true if arr1[0..n-1] and arr2[0..m-1]
// contain same elements.
bool areEqual(int arr1[], int arr2[], int n, int m)
{
    // If lengths of arrays are not equal
    if (n != m)
        return false;

    // Store arr1[] elements and their counts in
    // hash map
    unordered_map<int, int> mp;
    for (int i=0; i<n; i++)
        mp[arr1[i]]++;

    // Traverse arr2[] elements and check if all
    // elements of arr2[] are present same number
    // of times or not.
    for (int i=0; i<n; i++)
    {
       // If there is an element in arr2[], but
       // not in arr1[]
       if (mp.find(arr2[i]) == mp.end())
          return false;

       // If an element of arr2[] appears more
       // times than it appears in arr1[]
       if (mp[arr2[i]] == 0)
          return false;

        mp[arr2[i]]--;
    }

    return true;
}

// Driver Code
int main()
{
    int arr1[] = {3, 5, 2, 5, 2};
```

```
    int arr2[] = {2, 3, 5, 5, 2};
    int n = sizeof(arr1)/sizeof(int);
    int m = sizeof(arr2)/sizeof(int);

    if (areEqual(arr1, arr2, n, m))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java program to find given two array
// are equal or not using hashing technique
import java.util.*;
import java.io.*;

class GFG
{
    // Returns true if arr1[0..n-1] and arr2[0..m-1]
    // contain same elements.
    public static boolean areEqual(int arr1[], int arr2[])
    {
        int n = arr1.length;
        int m = arr2.length;

        // If lengths of arrays are not equal
        if (n != m)
            return false;

        // Store arr1[] elements and their counts in
        // hash map
        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        int count = 0;
        for (int i = 0; i < n; i++)
        {
            if(map.get(arr1[i]) == null)
                map.put(arr1[i], 1);
            else
            {
                count = map.get(arr1[i]);
                count ++;
                map.put(arr1[i], count);
            }
        }

        // Traverse arr2[] elements and check if all
```

```
        // elements of arr2[] are present same number
        // of times or not.
        for (int i = 0; i < n; i++)
        {
            // If there is an element in arr2[], but
            // not in arr1[]
            if (!map.containsKey(arr2[i]))
                return false;

            // If an element of arr2[] appears more
            // times than it appears in arr1[]
            if (map.get(arr2[i]) == 0)
                return false;

            count = map.get(arr2[i]);
            --count;
            map.put(arr2[i], count);
        }

        // again traverse arr2 to ensure that count
        // for all elelments become zero.
        for(int i = 0; i < n; i++)
        {
            if(map.get(arr2[i]) > 0)
                return false;
        }
        return true;
    }

    //Driver code
    public static void main (String[] args)
    {
        int arr1[] = { 3, 5, 2, 5, 2};
        int arr2[] = { 2, 3, 5, 5, 2};

        if (areEqual(arr1, arr2))
            System.out.println("Yes");
        else
            System.out.println("No");

    }
}
```

Output :

```
Yes
```

Time Complexity : O(n)

Auxiliary Space : O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/check-if-two-arrays-are-equal-or-not/

Chapter 22

Check if two strings are k-anagrams or not

Check if two strings are k-anagrams or not - GeeksforGeeks

Given two strings of lowercase alphabets and a value k, the task is to find if two strings are K-anagrams of each other or not.

Two strings are called **k-anagrams** if following two conditions are true.

1. Both have same number of characters.
2. Two strings can become anagram by changing at most k characters in a string.

Examples :

```
Input:  str1 = "anagram" , str2 = "grammar" , k = 3
Output:  Yes
Explanation: We can update maximum 3 values and
it can be done in changing only 'r' to 'n'
and 'm' to 'a' in str2.

Input:  str1 = "geeks", str2 = "eggkf", k = 1
Output:  No
Explanation: We can update or modify only 1
value but there is a need of modifying 2 characters.
i.e. g and f in str 2.
```

Below is a solution to check if two strings are k-anagrams of each other or not.

1. Stores occurrence of all characters of both strings in separate count arrays.

2. Count number of different characters in both strings (in this if a strings has 4 a and second has 3 'a' then it will be also count.
3. If count of different characters is less than or equal to k, then return true else false.

C++

```
 // C++ program to check if two strings are k anagram
// or not.
#include<bits/stdc++.h>
using namespace std;
const int MAX_CHAR = 26;

// Function to check that string is k-anagram or not
bool arekAnagrams(string str1, string str2, int k)
{
    // If both strings are not of equal
    // length then return false
    int n = str1.length();
    if (str2.length() != n)
        return false;

    int count1[MAX_CHAR] = {0};
    int count2[MAX_CHAR] = {0};

    // Store the occurrence of all characters
    // in a hash_array
    for (int i = 0; i < n; i++)
        count1[str1[i]-'a']++;
    for (int i = 0; i < n; i++)
        count2[str2[i]-'a']++;

    int count = 0;

    // Count number of characters that are
    // different in both strings
    for (int i = 0; i < MAX_CHAR; i++)
        if (count1[i] > count2[i])
            count = count + abs(count1[i]-count2[i]);

    // Return true if count is less than or
    // equal to k
    return (count <= k);
}

// Driver code
int main()
{
    string str1 = "anagram";
```

```
    string str2 = "grammar";
    int k = 2;
    if (arekAnagrams(str1, str2, k))
        cout << "Yes";
    else
        cout<< "No";
    return 0;
}
```

Java

```
 // Java program to check if two strings are k anagram
// or not.
public class GFG {

    static final int MAX_CHAR = 26;

    // Function to check that string is k-anagram or not
    static boolean arekAnagrams(String str1, String str2,
                                                int k)
    {
        // If both strings are not of equal
        // length then return false
        int n = str1.length();
        if (str2.length() != n)
            return false;

        int[] count1 = new int[MAX_CHAR];
        int[] count2 = new int[MAX_CHAR];
        int count = 0;

        // Store the occurrence of all characters
        // in a hash_array
        for (int i = 0; i < n; i++)
            count1[str1.charAt(i) - 'a']++;
        for (int i = 0; i < n; i++)
            count2[str2.charAt(i) - 'a']++;

        // Count number of characters that are
        // different in both strings
        for (int i = 0; i < MAX_CHAR; i++)
            if (count1[i] > count2[i])
                count = count + Math.abs(count1[i] -
                                        count2[i]);

        // Return true if count is less than or
        // equal to k
        return (count <= k);
```

```
    }

    // Driver code
    public static void main(String args[])
    {
        String str1 = "anagram";
        String str2 = "grammar";
        int k = 2;
        if (arekAnagrams(str1, str2, k))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
// This code is contributed by Sumit Ghosh
```

C#

```
 // C# program to check if two
// strings are k anagram or not.
using System;
class GFG {

    static int MAX_CHAR = 26;

    // Function to check that
    // string is k-anagram or not
    static bool arekAnagrams(string str1,
                             string str2,
                                      int k)
    {

        // If both strings are not of equal
        // length then return false
        int n = str1.Length;
        if (str2.Length != n)
            return false;

        int[] count1 = new int[MAX_CHAR];
        int[] count2 = new int[MAX_CHAR];
        int count = 0;

        // Store the occurrence
        // of all characters
        // in a hash_array
        for (int i = 0; i < n; i++)
            count1[str1[i] - 'a']++;
        for (int i = 0; i < n; i++)
```

```
            count2[str2[i] - 'a']++;

        // Count number of characters that are
        // different in both strings
        for (int i = 0; i < MAX_CHAR; i++)
            if (count1[i] > count2[i])
                count = count + Math.Abs(count1[i] -
                                         count2[i]);

        // Return true if count is
        // less than or equal to k
        return (count <= k);
    }

    // Driver code
    public static void Main()
    {
        string str1 = "anagram";
        string str2 = "grammar";
        int k = 2;
        if (arekAnagrams(str1, str2, k))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}
// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to check
// if two strings are
// k anagram or not.
$MAX_CHAR = 26;

// Function to check that
// string is k-anagram or not
function arekAnagrams($str1, $str2, $k)
{
    global $MAX_CHAR;
    // If both strings are not of
    // equal length then return false
    $n = strlen($str1);
    if (strlen($str2) != $n)
        return false;

    $count1 = (0);
```

```
    $count2 = (0);

    // Store the occurrence of all
    // characters in a hash_array
    $count = 0;

    // Count number of characters that
    // are different in both strings
    for ($i = 0; $i < $MAX_CHAR; $i++)
        if ($count1[$i] > $count2[$i])
            $count = $count + abs($count1[$i] -
                                  $count2[$i]);

    // Return true if count is
    // less than or equal to k
    return ($count <= $k);
}

// Driver Code
$str1 = "anagram";
$str2 = "grammar";
$k = 2;
if (arekAnagrams($str1, $str2, $k))
    echo "Yes";
else
    echo "No";

// This code is contributed by m_kit
?>
```

Output :

```
Yes
```

We can optimize above solution. Here we use only one count array to store counts of characters in str1. We traverse str2 and decrement occurrence of every character in count array that is present in str2. If we find a character that is not there in str1, we increment count of different characters. If count of different character become more than k, we return false.

C++

```
 // Optimized C++ program to check if two strings
// are k anagram or not.
#include<bits/stdc++.h>
using namespace std;
```

```
const int MAX_CHAR = 26;

// Function to check if str1 and str2 are k-anagram
// or not
bool areKAnagrams(string str1, string str2, int k)
{
    // If both strings are not of equal
    // length then return false
    int n = str1.length();
    if (str2.length() != n)
        return false;

    int hash_str1[MAX_CHAR] = {0};

    // Store the occurrence of all characters
    // in a hash_array
    for (int i = 0; i < n ; i++)
        hash_str1[str1[i]-'a']++;

    // Store the occurrence of all characters
    // in a hash_array
    int count = 0;
    for (int i = 0; i < n ; i++)
    {
        if (hash_str1[str2[i]-'a'] > 0)
            hash_str1[str2[i]-'a']--;
        else
            count++;

        if (count > k)
            return false;
    }

    // Return true if count is less than or
    // equal to k
    return true;
}

// Driver code
int main()
{
    string str1 = "fodr";
    string str2 = "gork";
    int k = 2;
    if (areKAnagrams(str1, str2, k) == true)
        cout << "Yes";
    else
        cout << "No";
```

```
    return 0;
}
```

Java

```
 // Optimized Java program to check if two strings
// are k anagram or not.
public class GFG {

    static final int MAX_CHAR = 26;

    // Function to check if str1 and str2 are k-anagram
    // or not
    static boolean areKAnagrams(String str1, String str2,
                                                int k)
    {
        // If both strings are not of equal
        // length then return false
        int n = str1.length();
        if (str2.length() != n)
            return false;

        int[] hash_str1 = new int[MAX_CHAR];

        // Store the occurrence of all characters
        // in a hash_array
        for (int i = 0; i < n ; i++)
            hash_str1[str1.charAt(i)-'a']++;

        // Store the occurrence of all characters
        // in a hash_array
        int count = 0;
        for (int i = 0; i < n ; i++)
        {
            if (hash_str1[str2.charAt(i)-'a'] > 0)
                hash_str1[str2.charAt(i)-'a']--;
            else
                count++;

            if (count > k)
                return false;
        }

        // Return true if count is less than or
        // equal to k
        return true;
    }
```

```
    // Driver code
    public static void main(String args[])
    {
        String str1 = "fodr";
        String str2 = "gork";
        int k = 2;
        if (areKAnagrams(str1, str2, k) == true)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
// This code is contributed by Sumit Ghosh
```

C#

```
 // Optimized C# program to check if two strings
// are k anagram or not.
using System;

 class GFG {

    static  int MAX_CHAR = 26;

    // Function to check if str1 and str2 are k-anagram
    // or not
    static bool areKAnagrams(String str1, String str2,
                                                int k)
    {
        // If both strings are not of equal
        // [i] then return false
        int n = str1.Length;
        if (str2.Length != n)
            return false;

        int[] hash_str1 = new int[MAX_CHAR];

        // Store the occurrence of all characters
        // in a hash_array
        for (int i = 0; i < n ; i++)
            hash_str1[str1[i]-'a']++;

        // Store the occurrence of all characters
        // in a hash_array
        int count = 0;
        for (int i = 0; i < n ; i++)
        {
            if (hash_str1[str2[i]-'a'] > 0)
```

```
                hash_str1[str2[i]-'a']--;
            else
                count++;

            if (count > k)
                return false;
        }

        // Return true if count is less than or
        // equal to k
        return true;
    }

    // Driver code
     static void Main()
    {
        String str1 = "fodr";
        String str2 = "gork";
        int k = 2;

        if (areKAnagrams(str1, str2, k) == true)
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}
// This code is contributed by Anuj_67
```

PHP

```
 <?php
// Optimized PHP program
// to check if two strings
// are k anagram or not.
$MAX_CHAR = 26;

// Function to check if str1
// and str2 are k-anagram or not
function areKAnagrams($str1,
                      $str2, $k)
{
    global $MAX_CHAR;
    // If both strings are
    // not of equal length
    // then return false

    $n = strlen($str1);
    if (strlen($str2) != $n)
```

```
        return false;

    $hash_str1 = array(0);

    // Store the occurrence of
    // all characters in a hash_array
    for ($i = 0; $i < $n ; $i++)
        $hash_str1[$str1[$i] - 'a']++;

    // Store the occurrence of all
    // characters in a hash_array
    $count = 0;
    for ($i = 0; $i < $n ; $i++)
    {
        if ($hash_str1[$str2[$i] - 'a'] > 0)
            $hash_str1[$str2[$i] - 'a']--;
        else
            $count++;

        if ($count > $k)
            return false;
    }

    // Return true if count is
    // less than or equal to k
    return true;
}

// Driver code
$str1 = "fodr";
$str2 = "gork";
$k = 2;
if (areKAnagrams($str1, $str2, $k) == true)
    echo "Yes";
else
    echo "No";

// This code is contributed by ajit
?>
```

Output:

```
Yes
```

Improved By : nitin mittal, vt_m, jit_t

Source

https://www.geeksforgeeks.org/check-two-strings-k-anagrams-not/

Chapter 23

Check whether Arithmetic Progression can be formed from the given array

Check whether Arithmetic Progression can be formed from the given array - GeeksforGeeks

Given an array of **n** integers. The task is to check whether an arithmetic progression can be formed using all the given elements. If possible print "Yes", else print "No".

Examples:

```
Input : arr[] = {0, 12, 4, 8}
Output : Yes
Rearrange given array as {0, 4, 8, 12}
which forms an arithmetic progression.

Input : arr[] = {12, 40, 11, 20}
Output : No
```

Method 1 (Simple)
A simple solution is to first find the smallest element, then find second smallest element and find the difference between these two. Let this difference be d. After finding the difference, find third smallest, fourth smallest and so on. After finding every i-th smallest smallest (from third onward), find the difference between value of current element and value of previous element. If difference is not same as d, return false. If all elements have same difference, return true. Time complexity of this solution is $O(n^2)$

Method 2(Use Sorting)
The idea is to sort the given array. After sorting, check if differences between consecutive elements are same or not. If all differences are same, Arithmetic Progression is possible.

Below is the implementation of this approach:

C++

```
 // C++ program to check if a given array
// can form arithmetic progression
#include<bits/stdc++.h>
using namespace std;

// Returns true if a permutation of arr[0..n-1]
// can form arithmetic progression
bool checkIsAP(int arr[], int n)
{
  if (n == 1)
    return true;

  // Sort array
  sort(arr, arr + n);

  // After sorting, difference between
  // consecutive elements must be same.
  int d = arr[1] - arr[0];
  for (int i=2; i<n; i++)
    if (arr[i] - arr[i-1] != d)
      return false;

  return true;
}

// Driven Program
int main()
{
  int arr[] = { 20, 15, 5, 0, 10 };
  int n = sizeof(arr)/sizeof(arr[0]);

  (checkIsAP(arr, n))? (cout << "Yes" << endl) :
                       (cout << "No" << endl);

  return 0;
}
```

Java

```
 // Java program to check if a given array
// can form arithmetic progression
import java.util.Arrays;
```

```
class GFG {

    // Returns true if a permutation of
    // arr[0..n-1] can form arithmetic
    // progression
    static boolean checkIsAP(int arr[], int n)
    {
        if (n == 1)
            return true;

        // Sort array
        Arrays.sort(arr);

        // After sorting, difference between
        // consecutive elements must be same.
        int d = arr[1] - arr[0];
        for (int i = 2; i < n; i++)
            if (arr[i] - arr[i-1] != d)
                return false;

        return true;
    }

    //driver code
    public static void main (String[] args)
    {
        int arr[] = { 20, 15, 5, 0, 10 };
        int n = arr.length;

        if(checkIsAP(arr, n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to check if a given
# array can form arithmetic progression

# Returns true if a permutation of arr[0..n-1]
# can form arithmetic progression
def checkIsAP(arr, n):
    if (n == 1): return True
```

```
    # Sort array
    arr.sort()

    # After sorting, difference between
    # consecutive elements must be same.
    d = arr[1] - arr[0]
    for i in range(2, n):
        if (arr[i] - arr[i-1] != d):
            return False

    return True

# Driver code
arr = [ 20, 15, 5, 0, 10 ]
n = len(arr)
print("Yes") if(checkIsAP(arr, n)) else print("No")

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to check if a given array
// can form arithmetic progression
using System;

class GFG {

    // Returns true if a permutation of
    // arr[0..n-1] can form arithmetic
    // progression
    static bool checkIsAP(int []arr, int n)
    {
        if (n == 1)
            return true;

        // Sort array
        Array.Sort(arr);

        // After sorting, difference between
        // consecutive elements must be same.
        int d = arr[1] - arr[0];
        for (int i = 2; i < n; i++)
            if (arr[i] - arr[i - 1] != d)
                return false;

        return true;
    }
```

```
    //Driver Code
    public static void Main ()
    {
        int []arr = {20, 15, 5, 0, 10};
        int n = arr.Length;

        if(checkIsAP(arr, n))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to check if
// a given array can form
// arithmetic progression

// Returns true if a permutation
// of arr[0..n-1] can form
// arithmetic progression
function checkIsAP($arr, $n)
{
    if ($n == 1)
        return true;

    // Sort array
    sort($arr);

    // After sorting, difference
    // between consecutive elements
    // must be same.
    $d = $arr[1] - $arr[0];
    for ($i = 2; $i < $n; $i++)
        if ($arr[$i] -
            $arr[$i - 1] != $d)
        return false;

    return true;
}

// Driver Code
$arr = array(20, 15, 5, 0, 10);
$n = count($arr);
```

```
if(checkIsAP($arr, $n))
echo "Yes";
else
echo "No";

// This code is contributed
// by Sam007
?>
```

Output:

```
Yes
```

Time Complexity: O(n Log n).

Method 3(Use Hashing)

1. Find out the smallest and second smallest elements
2. Find different between the two elements. d = second_smallest – smallest
3. Store all elements in a hashmap and return "NO" if duplicate element found (can be done together with step 1).
4. Now start from "second smallest element + d" and one by one check n-2 terms of Arithmetic Progression in hashmap. If any value of progression is missing, return false.
5. Return "YES" after end of the loop.

Time Complexity : O(n)
Auxiliary Space : O(n)

Thanks to **Chenna Rao** for suggesting this method,

Method 4(Using counting sort)
We can reduce space required in method 3 if given array can be modified.

1. Find smallest and second smallest elements.
2. Find d = second_smallest – smallest
3. Subtract smallest element from all elements.
4. Now if given array represent AP, all elements should be of form i*d where i varies from 0 to n-1.
5. One by one divide all reduced elements with d. If any element is not divisible by d, return false.
6. Now if array represents AP, it must be a permutation of numbers from 0 to n-1. We can easily check this using counting sort.

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/check-whether-arithmetic-progression-can-formed-given-array/

Chapter 24

Check whether BST contains Dead End or not

Check whether BST contains Dead End or not - GeeksforGeeks

Given a Binary search Tree that contains positive integer values greater then 0. The task is to check whether the BST contains a dead end or not. Here Dead End means, we are not able to insert any element after that node.

Examples:

```
Input :        8
             /   \
           5      9
         /   \
        2     7
       /
      1
Output : Yes
Explanation : Node "1" is the dead End because
         after that we cant insert any element.

Input :       8
            /   \
           7     10
         /      /   \
        2      9     13

Output : Yes
Explanation : We can't insert any element at
              node 9.
```

If we take a closer look at problem, we can notice that we basically need to check if there is leaf node with value x such that x+1 and x-1 exist in BST with exception of x = 1. For x = 1, we can't insert 0 as problem statement says BST contains positive integers only.

To implement above idea we first traverse whole BST and store all nodes in a hash_map. We also store all leaves in a separate hash to avoid re-traversal of BST. Finally we check for every leaf node x, if x-1 and x+1 are present in hash_map or not.

Below is C++ implementation of above idea .

```
 // C++ program check weather BST contains
// dead end or not
#include<bits/stdc++.h>
using namespace std;

// A BST node
struct Node
{
    int data;
    struct Node *left, *right;
};

// A utility function to create a new node
Node *newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new Node
  with given key in BST */
struct Node* insert(struct Node* node, int key)
{
    /* If the tree is empty, return a new Node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->data)
        node->left = insert(node->left, key);
    else if (key > node->data)
        node->right = insert(node->right, key);

    /* return the (unchanged) Node pointer */
    return node;
}

// Function to store all node of given binary search tree
```

```
void storeNodes(Node * root, unordered_set<int> &all_nodes,
                          unordered_set<int> &leaf_nodes)
{
    if (root == NULL)
        return ;

    // store all node of binary search tree
    all_nodes.insert(root->data);

    // store leaf node in leaf_hash
    if (root->left==NULL && root->right==NULL)
    {
        leaf_nodes.insert(root->data);
        return ;
    }

    // recur call rest tree
    storeNodes(root-> left, all_nodes, leaf_nodes);
    storeNodes(root->right, all_nodes, leaf_nodes);
}

// Returns true if there is a dead end in tree,
// else false.
bool isDeadEnd(Node *root)
{
    // Base case
    if (root == NULL)
        return false ;

    // create two empty hash sets that store all
    // BST elements and leaf nodes respectively.
    unordered_set<int> all_nodes, leaf_nodes;

    // insert 0 in 'all_nodes' for handle case
    // if bst contain value 1
    all_nodes.insert(0);

    // Call storeNodes function to store all BST Node
    storeNodes(root, all_nodes, leaf_nodes);

    // Traversal leaf node and check Tree contain
    // continuous sequence of
    // size tree or Not
    for (auto i = leaf_nodes.begin() ; i != leaf_nodes.end(); i++)
    {
        int x = (*i);

        // Here we check first and last element of
```

```
        // continuous sequence that are x-1 & x+1
        if (all_nodes.find(x+1) != all_nodes.end() &&
            all_nodes.find(x-1) != all_nodes.end())
            return true;
    }

    return false ;
}

// Driver program
int main()
{
/*       8
       /   \
      5    11
     /  \
    2    7
     \
      3
       \
        4 */
    Node *root = NULL;
    root = insert(root, 8);
    root = insert(root, 5);
    root = insert(root, 2);
    root = insert(root, 3);
    root = insert(root, 7);
    root = insert(root, 11);
    root = insert(root, 4);
    if (isDeadEnd(root) == true)
        cout << "Yes " << endl;
    else
        cout << "No " << endl;
    return 0;
}
```

Output:

```
Yes
```

Time Complexity : O(n)

Simple Recursive solution to check whether BST contains dead End

Source

https://www.geeksforgeeks.org/check-whether-bst-contains-dead-end-not/

Chapter 25

Clone a Binary Tree with Random Pointers

Clone a Binary Tree with Random Pointers - GeeksforGeeks

Given a Binary Tree where every node has following structure.

```
struct node {
    int key;
    struct node *left,*right,*random;
}
```

The random pointer points to any random node of the binary tree and can even point to NULL, clone the given binary tree.

Method 1 (Use Hashing)
The idea is to store mapping from given tree nodes to clone tre node in hashtable. Following are detailed steps.

1) Recursively traverse the given Binary and copy key value, left pointer and right pointer to clone tree. While copying, store the mapping from given tree node to clone tree node in a hashtable. In the following pseudo code, 'cloneNode' is currently visited node of clone tree and 'treeNode' is currently visited node of given tree.

```
cloneNode->key  = treeNode->key
cloneNode->left = treeNode->left
cloneNode->right = treeNode->right
map[treeNode] = cloneNode
```

2) Recursively traverse both trees and set random pointers using entries from hash table.

```
   cloneNode->random = map[treeNode->random]
```

Following is C++ implementation of above idea. The following implementation uses mapfrom C++ STL. Note that map doesn't implement hash table, it actually is based on self-balancing binary search tree.

```
 // A hashmap based C++ program to clone a binary tree with random pointers
#include<iostream>
#include<map>
using namespace std;

/* A binary tree node has data, pointer to left child, a pointer to right
   child and a pointer to random node*/
struct Node
{
    int key;
    struct Node* left, *right, *random;
};

/* Helper function that allocates a new Node with the
   given data and NULL left, right and random pointers. */
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->random = temp->right = temp->left = NULL;
    return (temp);
}

/* Given a binary tree, print its Nodes in inorder*/
void printInorder(Node* node)
{
    if (node == NULL)
        return;

    /* First recur on left sutree */
    printInorder(node->left);

    /* then print data of Node and its random */
    cout << "[" << node->key << " ";
    if (node->random == NULL)
        cout << "NULL], ";
    else
        cout << node->random->key << "], ";

    /* now recur on right subtree */
    printInorder(node->right);
```

```
}

// This function creates clone by copying key and left and right pointers
// This function also stores mapping from given tree node to clone.
Node* copyLeftRightNode(Node* treeNode, map<Node *, Node *> *mymap)
{
    if (treeNode == NULL)
        return NULL;
    Node* cloneNode = newNode(treeNode->key);
    (*mymap)[treeNode] = cloneNode;
    cloneNode->left  = copyLeftRightNode(treeNode->left, mymap);
    cloneNode->right = copyLeftRightNode(treeNode->right, mymap);
    return cloneNode;
}

// This function copies random node by using the hashmap built by
// copyLeftRightNode()
void copyRandom(Node* treeNode,  Node* cloneNode, map<Node *, Node *> *mymap)
{
    if (cloneNode == NULL)
        return;
    cloneNode->random =  (*mymap)[treeNode->random];
    copyRandom(treeNode->left, cloneNode->left, mymap);
    copyRandom(treeNode->right, cloneNode->right, mymap);
}

// This function makes the clone of given tree. It mainly uses
// copyLeftRightNode() and copyRandom()
Node* cloneTree(Node* tree)
{
    if (tree == NULL)
        return NULL;
    map<Node *, Node *> *mymap = new  map<Node *, Node *>;
    Node* newTree = copyLeftRightNode(tree, mymap);
    copyRandom(tree, newTree, mymap);
    return newTree;
}

/* Driver program to test above functions*/
int main()
{
    //Test No 1
    Node *tree = newNode(1);
    tree->left = newNode(2);
    tree->right = newNode(3);
    tree->left->left = newNode(4);
    tree->left->right = newNode(5);
    tree->random = tree->left->right;
```

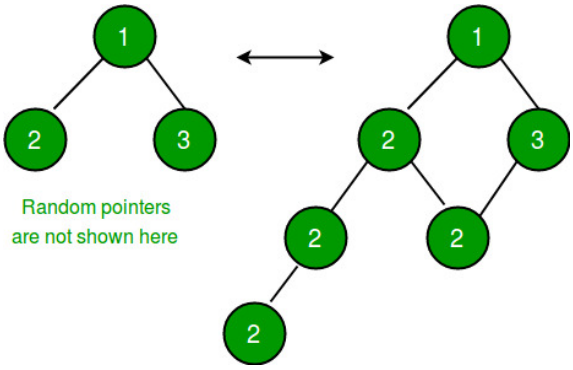
```
    tree->left->left->random = tree;
    tree->left->right->random = tree->right;

    //  Test No 2
    //    tree = NULL;

    //  Test No 3
    //    tree = newNode(1);

    //  Test No 4
    /*    tree = newNode(1);
        tree->left = newNode(2);
        tree->right = newNode(3);
        tree->random = tree->right;
        tree->left->random = tree;
    */

    cout << "Inorder traversal of original binary tree is: \n";
    printInorder(tree);

    Node *clone = cloneTree(tree);

    cout << "\n\nInorder traversal of cloned binary tree is: \n";
    printInorder(clone);

    return 0;
}
```

Output:

```
Inorder traversal of original binary tree is:
[4 1], [2 NULL], [5 3], [1 5], [3 NULL],

Inorder traversal of cloned binary tree is:
[4 1], [2 NULL], [5 3], [1 5], [3 NULL],
```

Method 2 (Temporarily Modify the Given Binary Tree)

1. Create new nodes in cloned tree and insert each new node in original tree between the left pointer edge of corresponding node in the original tree (See the below image).
i.e. if current node is A and it's left child is B (A — >> B), then new cloned node with key A wil be created (say cA) and it will be put as A — >> cA — >> B (B can be a NULL or a non-NULL left child). Right child pointer will be set correctly i.e. if for current node A, right child is C in original tree (A — >> C) then corresponding cloned nodes cA and cC will like cA —- >> cC



2. Set random pointer in cloned tree as per original tree
i.e. if node A's random pointer points to node B, then in cloned tree, cA will point to cB (cA and cB are new node in cloned tree corresponding to node A and B in original tree)

3. Restore left pointers correctly in both original and cloned tree

Following is C++ implementation of above algorithm.

```
 #include <iostream>
using namespace std;

/* A binary tree node has data, pointer to left child, a pointer to right
   child and a pointer to random node*/
struct Node
{
    int key;
    struct Node* left, *right, *random;
};

/* Helper function that allocates a new Node with the
   given data and NULL left, right and random pointers. */
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->random = temp->right = temp->left = NULL;
    return (temp);
}

/* Given a binary tree, print its Nodes in inorder*/
void printInorder(Node* node)
{
    if (node == NULL)
        return;

    /* First recur on left sutree */
    printInorder(node->left);
```

```
    /* then print data of Node and its random */
    cout << "[" << node->key << " ";
    if (node->random == NULL)
        cout << "NULL], ";
    else
        cout << node->random->key << "], ";

    /* now recur on right subtree */
    printInorder(node->right);
}

// This function creates new nodes cloned tree and puts new cloned node
// in between current node and it's left child
// i.e. if current node is A and it's left child is B ( A --- >> B ),
//      then new cloned node with key A wil be created (say cA) and
//      it will be put as
//      A --- >> cA --- >> B
// Here B can be a NULL or a non-NULL left child
// Right child pointer will be set correctly
// i.e. if for current node A, right child is C in original tree
// (A --- >> C) then corresponding cloned nodes cA and cC will like
// cA ---- >> cC
Node* copyLeftRightNode(Node* treeNode)
{
    if (treeNode == NULL)
        return NULL;

    Node* left = treeNode->left;
    treeNode->left = newNode(treeNode->key);
    treeNode->left->left = left;
    if(left != NULL)
        left->left = copyLeftRightNode(left);

    treeNode->left->right = copyLeftRightNode(treeNode->right);
    return treeNode->left;
}

// This function sets random pointer in cloned tree as per original tree
// i.e. if node A's random pointer points to node B, then
// in cloned tree, cA wil point to cB (cA and cB are new node in cloned
// tree corresponding to node A and B in original tree)
void copyRandomNode(Node* treeNode, Node* cloneNode)
{
    if (treeNode == NULL)
        return;
    if(treeNode->random != NULL)
        cloneNode->random = treeNode->random->left;
```

```
    else
        cloneNode->random = NULL;

    if(treeNode->left != NULL && cloneNode->left != NULL)
        copyRandomNode(treeNode->left->left, cloneNode->left->left);
    copyRandomNode(treeNode->right, cloneNode->right);
}

// This function will restore left pointers correctly in
// both original and cloned tree
void restoreTreeLeftNode(Node* treeNode, Node* cloneNode)
{
    if (treeNode == NULL)
        return;
    if (cloneNode->left != NULL)
    {
        Node* cloneLeft = cloneNode->left->left;
        treeNode->left = treeNode->left->left;
        cloneNode->left = cloneLeft;
    }
    else
        treeNode->left = NULL;

    restoreTreeLeftNode(treeNode->left, cloneNode->left);
    restoreTreeLeftNode(treeNode->right, cloneNode->right);
}

//This function makes the clone of given tree
Node* cloneTree(Node* treeNode)
{
    if (treeNode == NULL)
        return NULL;
    Node* cloneNode = copyLeftRightNode(treeNode);
    copyRandomNode(treeNode, cloneNode);
    restoreTreeLeftNode(treeNode, cloneNode);
    return cloneNode;
}


/* Driver program to test above functions*/
int main()
{
/*  //Test No 1
    Node *tree = newNode(1);
    tree->left = newNode(2);
    tree->right = newNode(3);
    tree->left->left = newNode(4);
    tree->left->right = newNode(5);
```

```
    tree->random = tree->left->right;
    tree->left->left->random = tree;
    tree->left->right->random = tree->right;

//  Test No 2
//    Node *tree = NULL;
/*
//  Test No 3
    Node *tree = newNode(1);

//  Test No 4
    Node *tree = newNode(1);
    tree->left = newNode(2);
    tree->right = newNode(3);
    tree->random = tree->right;
    tree->left->random = tree;

  Test No 5
    Node *tree = newNode(1);
    tree->left = newNode(2);
    tree->right = newNode(3);
    tree->left->left = newNode(4);
    tree->left->right = newNode(5);
    tree->right->left = newNode(6);
    tree->right->right = newNode(7);
    tree->random = tree->left;
*/
//    Test No 6
    Node *tree = newNode(10);
    Node *n2 = newNode(6);
    Node *n3 = newNode(12);
    Node *n4 = newNode(5);
    Node *n5 = newNode(8);
    Node *n6 = newNode(11);
    Node *n7 = newNode(13);
    Node *n8 = newNode(7);
    Node *n9 = newNode(9);
    tree->left = n2;
    tree->right = n3;
    tree->random = n2;
    n2->left = n4;
    n2->right = n5;
    n2->random = n8;
    n3->left = n6;
    n3->right = n7;
    n3->random = n5;
    n4->random = n9;
    n5->left = n8;
```

```
    n5->right = n9;
    n5->random = tree;
    n6->random = n9;
    n9->random = n8;

/*    Test No 7
    Node *tree = newNode(1);
    tree->left = newNode(2);
    tree->right = newNode(3);
    tree->left->random = tree;
    tree->right->random = tree->left;
*/
    cout << "Inorder traversal of original binary tree is: \n";
    printInorder(tree);

    Node *clone = cloneTree(tree);

    cout << "\n\nInorder traversal of cloned binary tree is: \n";
    printInorder(clone);

    return 0;
}
```

Output:

```
Inorder traversal of original binary tree is:
[5 9], [6 7], [7 NULL], [8 10], [9 7], [10 6], [11 9], [12 8], [13 NULL],

Inorder traversal of cloned binary tree is:
[5 9], [6 7], [7 NULL], [8 10], [9 7], [10 6], [11 9], [12 8], [13 NULL],
```

This article is contributed by **Anurag Singh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/clone-binary-tree-random-pointers/

Chapter 26

Coalesced hashing

Coalesced hashing - GeeksforGeeks

Coalesced hashing is a collision avoidance technique when there is a fixed sized data. It is a combination of both Separate chaining and Open addressing. It uses the concept of Open Addressing(linear probing) to find first empty place for colliding element from the bottom of the hash table and the concept of Separate Chaining to link the colliding elements to each other through pointers. The hash function used is **h=(key)%(total number of keys)**. Inside the hash table, each node has three fields:

- h(key): The value of hash function for a key.
- Data: The key itself.
- Next: The link to the next colliding elements.

The basic operations of Coalesced hashing are:

1. **INSERT(key):** The insert Operation inserts the key according to the hash value of that key if that hash value in the table is empty otherwise the key is inserted in first empty place from the bottom of the hash table and the address of this empty place is mapped in NEXT field of the previous pointing node of the chain.(Explained in example below).
2. **DELETE(Key):** The key if present is deleted.Also if the node to be deleted contains the address of another node in hash table then this address is mapped in the NEXT field of the node pointing to the node which is to be deleted
3. **SEARCH(key):** Returns **True** if key is present, otherwise return **False.**

The best case complexity of all these operations is O(1) and the worst case complexity is O(n) where n is the total number of keys.It is better than separate chaining because it inserts the colliding element in the memory of hash table only instead of creating a new linked list as in separate chaining.
Illustration:
Example:

```
n = 10
Input : {20, 35, 16, 40, 45, 25, 32, 37, 22, 55}
```

Hash function

```
h(key) = key%10
```

Steps:

1. Initially empty hash table is created with all NEXT field initialised with NULL and h(key) values ranging from 0-9.
2. Let's start with inserting 20, as h(20)=0 and 0 index is empty so we insert 20 at 0 index.
3. Next element to be inserted is 35, h(35)=5 and 5th index empty so we insert 35 there.
4. Next we have 16, h(16)=6 which is empty so 16 is inserted at 6 index value.
5. Now we have to insert 40, h(40)=0 which is already occupied so we search for the first empty block from the bottom and insert it there i.e 9 index value.Also the address of this newly inserted node(from address we mean index value of a node) i.e(9)is initialised in the next field of 0th index value node.
6. To insert 45, h(45)=5 which is occupied so again we search for the empty block from the bottom i.e 8 index value and map the address of this newly inserted node i.e(8) to the Next field of 5th index value node i.e in the next field of key=35.
7. Next to insert 25, h(25)=5 is occupied so search for the first empty block from bottom i.e 7th index value and insert 25 there. Now it is important to note that the address of this new node cant be mapped on 5th index value node which is already pointing to some other node. To insert the address of new node we have to follow the link chain from the 5th index node until we get NULL in next field and map the address of new node to next field of that node i.e from 5th index node we go to 8th index node which contains NULL in next field so we insert address of new node i.e(7) in next field of 8th index node.
8. To insert 32, h(32)=2, which is empty so insert 32 at 2nd index value.
9. To insert 37, h(37)=7 which is occupied so search for the first free block from bottom which is 4th index value. So insert 37 at 4th index value and copy the address of this node in next field of 7th index value node.
10. To insert 22, h(22)=2 which is occupied so insert it at 3rd index value and map the address of this node in next field of 2nd index value node.
11. Finally, to insert 55 h(55)=5 which is occupied and the only empty space is 1st index value so insert 55 there. Now again to map the address of this new node we have to follow the chain starting from 5th index value node until we get NULL in next field i.e from 5th index->8th index->7th index->4th index which contains NULL in Next fied, and we insert the address of newly inserted node at 4th index value node.

Hash value	Data	Next
0	20	9

Hash value	Data	Next
1	55	NULL
2	32	3
3	22	NULL
4	37	1
5	35	8
6	16	NULL
7	25	4
8	45	7
9	40	NULL

Deletion process is simple, for example:
Case 1: To delete key=37, first search for 37. If it is present then simply delete the data value and if the node contains any address in next field and the node to be deleted i.e 37 is itself pointed by some other node(i.e key=25) then copy that address in the next field of 37 to the next field of node pointing to 37(i.e key=25) and initialize the NEXT field of key=37 as NULL again and erase the key=37.

Hash value	Data	Next
0	20	9
1	55	NULL
2	32	3
3	22	NULL
4	–	NULL
5	35	8
6	16	NULL
7	25	1
8	45	7
9	40	NULL

Case 2: If key to be deleted is 35 which is not pointed by any other node then we have to pull the chain attached to the node to be deleted i.e 35 one step back and mark last value of chain to NULL again.

Hash value	Data	Next
0	20	9
1	–	NULL
2	32	3
3	22	NULL
4	–	NULL
5	45	8
6	16	NULL
7	55	NULL
8	25	7

Hash value	Data	Next
9	40	NULL

Source

https://www.geeksforgeeks.org/coalesced-hashing/

Chapter 27

Common elements in all rows of a given matrix

Common elements in all rows of a given matrix - GeeksforGeeks

Given an m x n matrix, find all common elements present in all rows in O(mn) time and one traversal of matrix.

Example:

```
Input:
mat[4][5] = {{1, 2, 1, 4, 8},
             {3, 7, 8, 5, 1},
             {8, 7, 7, 3, 1},
             {8, 1, 2, 7, 9},
            };

Output:
1 8 or 8 1
8 and 1 are present in all rows.
```

A **simple solution** is to consider every element and check if it is present in all rows. If present, then print it.

A **better solution** is to sort all rows in the matrix and use similar approach as discussed here. Sorting will take O(mnlogn) time and finding common elements will take O(mn) time. So overall time complexity of this solution is O(mnlogn)

Can we do better than O(mnlogn)?
The idea is to use maps. We initially insert all elements of the first row in an map. For every other element in remaining rows, we check if it is present in the map. If it is present

in the map and is not duplicated in current row, we increment count of the element in map by 1, else we ignore the element. If the currently traversed row is the last row, we print the element if it has appeared m-1 times before.

Below is C++ implementation of the idea.

```
 // A Program to prints common element in all
// rows of matrix
#include <iostream>
#include <unordered_map>
using namespace std;

// Specify number of rows and columns
#define M 4
#define N 5

// prints common element in all rows of matrix
void printCommonElements(int mat[M][N])
{
    unordered_map<int, int> mp;

    // initalize 1st row elements with value 1
    for (int j = 0; j < N; j++)
        mp[mat[0][j]] = 1;

    // traverse the matrix
    for (int i = 1; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            // If element is present in the map and
            // is not duplicated in current row.
            if (mp[mat[i][j]] == i)
            {
               // we increment count of the element
               // in map by 1
                mp[mat[i][j]] = i + 1;

                // If this is last row
                if (i==M-1)
                  cout << mat[i][j] << " ";
            }
        }
    }
}

// driver program to test above function
int main()
{
```

```
    int mat[M][N] =
    {
        {1, 2, 1, 4, 8},
        {3, 7, 8, 5, 1},
        {8, 7, 7, 3, 1},
        {8, 1, 2, 7, 9},
    };

    printCommonElements(mat);

    return 0;
}
```

Output:

```
8 1
```

The time complexity of this solution is O(m * n) and we are doing only one traversal of the matrix.

Source

https://www.geeksforgeeks.org/common-elements-in-all-rows-of-a-given-matrix/

Chapter 28

Concatenated string with uncommon characters of two strings

Concatenated string with uncommon characters of two strings - GeeksforGeeks

Two strings are given and you have to modify 1st string such that all the common characters of the 2nd strings have to be removed and the uncommon characters of the 2nd string have to be concatenated with uncommon characters of the 1st string.

Examples:

```
Input : S1 = "aacdb"
        S2 = "gafd"
Output : "cbgf"

Input : S1 = "abcs";
        S2 = "cxzca";
Output : "bsxz"
```

The idea is to use hash map where key is character and value is number of strings in which character is present. If a character is present in one string, then count is 1, else if character is present in both strings, count is 2.

1. Initialize result as empty string.
2. Push all characters of 2nd string in map with count as 1.
3. Traverse first string and append all those characters to result that are not present in map. Characters that are present in map, make count 2.
4. Traverse second string and append all those characters to result whose count is 1.

```
 // C++ program Find concatenated string with
// uncommon characters of given strings
#include <bits/stdc++.h>
using namespace std;

string concatenetedString(string s1, string s2)
{
    string res = ""; // result

    // store all characters of s2 in map
    unordered_map<char, int> m;
    for (int i = 0; i < s2.size(); i++)
        m[s2[i]] = 1;

    // Find characters of s1 that are not
    // present in s2 and append to result
    for (int i = 0; i < s1.size(); i++)
    {
        if (m.find(s1[i]) == m.end())
            res += s1[i];
        else
            m[s1[i]] = 2;
    }

    // Find characters of s2 that are not
    // present in s1.
    for (int i = 0; i < s2.size(); i++)
        if (m[s2[i]] == 1)
            res += s2[i];
    return res;
}

/* Driver program to test above function */
int main()
{
    string s1 = "abcs";
    string s2 = "cxzca";
    cout << concatenetedString(s1, s2);
    return 0;
}
```

Output:

```
bsxz
```

Asked In : Microsoft

Source

https://www.geeksforgeeks.org/concatenated-string-uncommon-characters-two-strings/

Chapter 29

Construct a Binary Tree from Postorder and Inorder

Construct a Binary Tree from Postorder and Inorder - GeeksforGeeks

Given Postorder and Inorder traversals, construct the tree.

Examples:

```
Input :
in[]   = {2, 1, 3}
post[] = {2, 3, 1}

Output : Root of below tree
      1
    /   \
   2     3


Input :
in[]   = {4, 8, 2, 5, 1, 6, 3, 7}
post[] = {8, 4, 5, 2, 6, 7, 3, 1}

Output : Root of below tree
          1
       /     \
     2         3
   /    \    /   \
  4     5   6    7
    \
      8
```

We have already discussed construction of tree from iven Inorder and Preorder traversals. The idea is similar.

Let us see the process of constructing tree from in[] = {4, 8, 2, 5, 1, 6, 3, 7} and post[] = {8, 4, 5, 2, 6, 7, 3, 1}

1) We first find the last node in post[]. The last node is "1", we know this value is root as root always appear in the end of postorder traversal.

2) We search "1" in in[] to find left and right subtrees of root. Everything on left of "1" in in[] is in left subtree and everything on right is in right subtree.

```
          1
        /    \
[4, 8, 2, 5]   [6, 3, 7]
```

3) We recur the above process for following two.
....**b)** Recur for in[] = {6, 3, 7} and post[] = {6, 7, 3}
.....Make the created tree as right child of root.
....**a)** Recur for in[] = {4, 8, 2, 5} and post[] = {8, 4, 5, 2}.
.....Make the created tree as left child of root.

Below is C++ implementation of above idea. One important observation is, we recursively call for right subtree before left subtree as we decrease index of postorder index whenever we create a new node.

C++

```
 /* C++ program to construct tree using inorder and
   postorder traversals */
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left
   child and a pointer to right child */
struct Node {
    int data;
    Node *left, *right;
};

// Utility function to create a new node
Node* newNode(int data);

/* Prototypes for utility functions */
int search(int arr[], int strt, int end, int value);

/* Recursive function to construct binary of size n
   from  Inorder traversal in[] and Postorder traversal
```

```
   post[].  Initial values of inStrt and inEnd should
   be 0 and n -1.  The function doesn't do any error
   checking for cases where inorder and postorder
   do not form a tree */
Node* buildUtil(int in[], int post[], int inStrt,
                int inEnd, int* pIndex)
{
    // Base case
    if (inStrt > inEnd)
        return NULL;

    /* Pick current node from Postorder traversal using
       postIndex and decrement postIndex */
    Node* node = newNode(post[*pIndex]);
    (*pIndex)--;

    /* If this node has no children then return */
    if (inStrt == inEnd)
        return node;

    /* Else find the index of this node in Inorder
       traversal */
    int iIndex = search(in, inStrt, inEnd, node->data);

    /* Using index in Inorder traversal, construct left and
       right subtress */
    node->right = buildUtil(in, post, iIndex + 1, inEnd, pIndex);
    node->left = buildUtil(in, post, inStrt, iIndex - 1, pIndex);

    return node;
}

// This function mainly initializes index of root
// and calls buildUtil()
Node* buildTree(int in[], int post[], int n)
{
    int pIndex = n - 1;
    return buildUtil(in, post, 0, n - 1, &pIndex);
}

/* Function to find index of value in arr[start...end]
   The function assumes that value is postsent in in[] */
int search(int arr[], int strt, int end, int value)
{
    int i;
    for (i = strt; i <= end; i++) {
        if (arr[i] == value)
            break;
```

```
    }
    return i;
}

/* Helper function that allocates a new node */
Node* newNode(int data)
{
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}

/* This funtcion is here just to test  */
void preOrder(Node* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preOrder(node->left);
    preOrder(node->right);
}

// Driver code
int main()
{
    int in[] = { 4, 8, 2, 5, 1, 6, 3, 7 };
    int post[] = { 8, 4, 5, 2, 6, 7, 3, 1 };
    int n = sizeof(in) / sizeof(in[0]);

    Node* root = buildTree(in, post, n);

    cout << "Preorder of the constructed tree : \n";
    preOrder(root);

    return 0;
}
```

Java

```
 // Java program to construct a tree using inorder
// and postorder traversals

/* A binary tree node has data, pointer to left
   child and a pointer to right child */
class Node {
    int data;
    Node left, right;
```

```
    public Node(int data)
    {
        this.data = data;
        left = right = null;
    }
}

// Class Index created to implement pass by reference of Index
class Index {
    int index;
}

class BinaryTree {
    /* Recursive function to construct binary of size n
       from  Inorder traversal in[] and Postrder traversal
       post[].  Initial values of inStrt and inEnd should
       be 0 and n -1.  The function doesn't do any error
       checking for cases where inorder and postorder
       do not form a tree */
    Node buildUtil(int in[], int post[], int inStrt,
                   int inEnd, Index pIndex)
    {
        // Base case
        if (inStrt > inEnd)
            return null;

        /* Pick current node from Postrder traversal using
           postIndex and decrement postIndex */
        Node node = new Node(post[pIndex.index]);
        (pIndex.index)--;

        /* If this node has no children then return */
        if (inStrt == inEnd)
            return node;

        /* Else find the index of this node in Inorder
           traversal */
        int iIndex = search(in, inStrt, inEnd, node.data);

        /* Using index in Inorder traversal, construct left and
           right subtress */
        node.right = buildUtil(in, post, iIndex + 1, inEnd, pIndex);
        node.left = buildUtil(in, post, inStrt, iIndex - 1, pIndex);

        return node;
    }
```

```
    // This function mainly initializes index of root
    // and calls buildUtil()
    Node buildTree(int in[], int post[], int n)
    {
        Index pIndex = new Index();
        pIndex.index = n - 1;
        return buildUtil(in, post, 0, n - 1, pIndex);
    }

    /* Function to find index of value in arr[start...end]
       The function assumes that value is postsent in in[] */
    int search(int arr[], int strt, int end, int value)
    {
        int i;
        for (i = strt; i <= end; i++) {
            if (arr[i] == value)
                break;
        }
        return i;
    }

    /* This funtcion is here just to test  */
    void preOrder(Node node)
    {
        if (node == null)
            return;
        System.out.print(node.data + " ");
        preOrder(node.left);
        preOrder(node.right);
    }

    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        int in[] = new int[] { 4, 8, 2, 5, 1, 6, 3, 7 };
        int post[] = new int[] { 8, 4, 5, 2, 6, 7, 3, 1 };
        int n = in.length;
        Node root = tree.buildTree(in, post, n);
        System.out.println("Preorder of the constructed tree : ");
        tree.preOrder(root);
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Output :

```
Preorder of the constructed tree :
1 2 4 8 5 3 6 7
```

Time Complexity : $O(n^2)$

Optimized approach: We can optimize the above solution using hashing (unordered_map in C++ or HashMap in Java). We store indexes of inorder traversal in a hash table. So that search can be done O(1) time.

```
 /* C++ program to construct tree using inorder and
postorder traversals */
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left
child and a pointer to right child */
struct Node {
    int data;
    Node *left, *right;
};

// Utility function to create a new node
Node* newNode(int data);

/* Recursive function to construct binary of size n
from Inorder traversal in[] and Postorder traversal
post[]. Initial values of inStrt and inEnd should
be 0 and n -1. The function doesn't do any error
checking for cases where inorder and postorder
do not form a tree */
Node* buildUtil(int in[], int post[], int inStrt,
    int inEnd, int* pIndex, unordered_map<int, int>& mp)
{
    // Base case
    if (inStrt > inEnd)
        return NULL;

    /* Pick current node from Postorder traversal
    using postIndex and decrement postIndex */
    int curr = post[*pIndex];
    Node* node = newNode(curr);
    (*pIndex)--;

    /* If this node has no children then return */
    if (inStrt == inEnd)
        return node;

    /* Else find the index of this node in Inorder
```

```
    traversal */
    int iIndex = mp[curr];

    /* Using index in Inorder traversal, construct
    left and right subtress */
    node->right = buildUtil(in, post, iIndex + 1,
                            inEnd, pIndex, mp);
    node->left = buildUtil(in, post, inStrt,
                           iIndex - 1, pIndex, mp);

    return node;
}

// This function mainly creates an unordered_map, then
// calls buildTreeUtil()
struct Node* buildTree(int in[], int post[], int len)
{
    // Store indexes of all items so that we
    // we can quickly find later
    unordered_map<int, int> mp;
    for (int i = 0; i < len; i++)
        mp[in[i]] = i;

    int index = len - 1; // Index in postorder
    return buildUtil(in, post, 0, len - 1,
                     &index, mp);
}

/* Helper function that allocates a new node */
Node* newNode(int data)
{
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}

/* This funtcion is here just to test */
void preOrder(Node* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preOrder(node->left);
    preOrder(node->right);
}

// Driver code
```

```
int main()
{
    int in[] = { 4, 8, 2, 5, 1, 6, 3, 7 };
    int post[] = { 8, 4, 5, 2, 6, 7, 3, 1 };
    int n = sizeof(in) / sizeof(in[0]);

    Node* root = buildTree(in, post, n);

    cout << "Preorder of the constructed tree : \n";
    preOrder(root);

    return 0;
}
```

Source

https://www.geeksforgeeks.org/construct-a-binary-tree-from-postorder-and-inorder/

Time Complexity : O(n)
This article is contributed by **Rishi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Chapter 30

Construct a tree from Inorder and Level order traversals | Set 2

Construct a tree from Inorder and Level order traversals | Set 2 - GeeksforGeeks

Given inorder and level-order traversals of a Binary Tree, construct the Binary Tree. Following is an example to illustrate the problem.

Examples:

```
Input: Two arrays that represent Inorder
       and level order traversals of a
       Binary Tree
in[]    = {4, 8, 10, 12, 14, 20, 22};
level[] = {20, 8, 22, 4, 12, 10, 14};

Output: Construct the tree represented
        by the two arrays.
        For the above two arrays, the
        constructed tree is shown.
```

We have discussed a solution in below post that works in O(N^3)
Construct a tree from Inorder and Level order traversals | Set 1

Approach : Following algorithm uses O(N^2) time complexity to solve the above problem using the unordered_set data structure in c++ (basically making a hash-table) to put the values of left subtree of the current root and later and we will check in O(1) complexity to find if the current levelOrder node is part of left subtree or not.

If it is the part of left subtree then add in one lLevel arrray for left other wise add it to rLevel array for right subtree.

Below is the c++ implementation with the above idea

```
 /* program to construct tree using inorder
   and levelorder traversals */
#include <iostream>
#include<set>
using namespace std;

/* A binary tree node */
struct Node
{
    int key;
    struct Node* left, *right;
};

Node* makeNode(int data){
    Node* newNode = new Node();
    newNode->key = data;
    newNode->right = newNode->right = NULL;
    return newNode;
}

// Function to build tree from given
// levelorder and inorder
Node* buildTree(int inorder[], int levelOrder[],
                int iStart, int iEnd, int n)
{
    if (n <= 0) r
          eturn NULL;

    // First node of level order is root
    Node* root = makeNode(levelOrder[0]);

    // Search root in inorder
    int index = -1;
    for (int i=iStart; i<=iEnd; i++){
        if (levelOrder[0] == inorder[i]){
            index = i;
            break;
        }
    }

    // Insert all left nodes in hash table
    unordered_set<int> s;
    for (int i=iStart;i<index;i++)
        s.insert(inorder[i]);
```

```
    // Separate level order traversals
    // of left and right subtrees.
    int lLevel[s.size()];  // Left
    int rLevel[iEnd-iStart-s.size()]; // Right
    int li = 0, ri = 0;
    for (int i=1;i<n;i++) {
        if (s.find(levelOrder[i]) != s.end())
            lLevel[li++] = levelOrder[i];
        else
            rLevel[ri++] = levelOrder[i];
    }

    // Recursively build left and right
    // subtrees and return root.
    root->left = buildTree(inorder, lLevel,
                 iStart, index-1, index-iStart);
    root->right = buildTree(inorder, rLevel,
                  index+1, iEnd, iEnd-index);
    return root;

}

/* Utility function to print inorder
traversal of binary tree */
void printInorder(Node* node)
{
    if (node == NULL)
       return;
    printInorder(node->left);
    cout << node->key << " ";
    printInorder(node->right);
}

// Driver Code
int main()
{
    int in[] = {4, 8, 10, 12, 14, 20, 22};
    int level[] = {20, 8, 22, 4, 12, 10, 14};
    int n = sizeof(in)/sizeof(in[0]);
    Node *root = buildTree(in, level, 0,
                           n - 1, n);

    /* Let us test the built tree by
     printing Insorder traversal */
    cout << "Inorder traversal of the "
            "constructed tree is \n";
    printInorder(root);
```

```
    return 0;
}
```

Output :

```
Inorder traversal of the
constructed tree is 4 8 10 12 14
20 22
```

Time Complexity: O(N^2)

Source

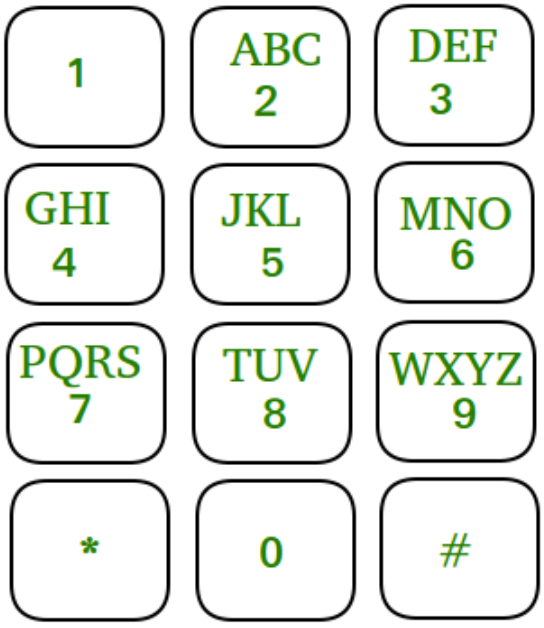
https://www.geeksforgeeks.org/construct-tree-inorder-level-order-traversals-set-2/

Chapter 31

Convert a sentence into its equivalent mobile numeric keypad sequence

Convert a sentence into its equivalent mobile numeric keypad sequence - GeeksforGeeks

Given a sentence in the form of a string, convert it into its equivalent mobile numeric keypad sequence.



Examples :

```
Input : GEEKSFORGEEKS
Output : 4333355777733366677743333557777
For obtaining a number, we need to press a
number corresponding to that character for
number of times equal to position of the
character. For example, for character C,
we press number 2 three times and accordingly.

Input : HELLO WORLD
Output : 44335555556660966677755 53
```

Follow the steps given below to convert a sentence into its equivalent mobile numeric keypad sequence.

- For each character, store the sequence which should be obtained at its respective position in an array, i.e. for Z, store 9999. For Y, store 999. For K, store 55 and so on.
- For each character, subtract ASCII value of 'A' and obtain the position in the array pointed
  by that character and add the sequence stored in that array to a string.
- If the character is a space, store 0
- Print the overall sequence.

Below is the implementation of above method :

C++

```
 // C++ implementation to convert a
// sentence into its equivalent
// mobile numeric keypad sequence
#include <bits/stdc++.h>
using namespace std;

// Function which computes the sequence
string printSequence(string arr[],
                     string input)
{
    string output = "";

    // length of input string
    int n = input.length();
    for (int i=0; i<n; i++)
    {
        // Checking for space
        if (input[i] == ' ')
            output = output + "0";
```

```
        else
        {
            // Calculating index for each
            // character
            int position = input[i]-'A';
            output = output + arr[position];
        }
    }

    // Output sequence
    return output;
}

// Driver function
int main()
{
    // storing the sequence in array
    string str[] = {"2","22","222",
                    "3","33","333",
                    "4","44","444",
                    "5","55","555",
                    "6","66","666",
                    "7","77","777","7777",
                    "8","88","888",
                    "9","99","999","9999"
                   };

    string input = "GEEKSFORGEEKS";
    cout << printSequence(str, input);
    return 0;
}
```

Java

```
 // Java implementation to convert a
// sentence into its equivalent
// mobile numeric keypad sequence
import java.util.*;

class GFG
{

    // Function which computes the sequence
    static String printSequence(String arr[],
                                String input)
    {
        String output = "";
```

```
        // length of input string
        int n = input.length();
        for (int i = 0; i < n; i++)
        {
            // Checking for space
            if (input.charAt(i) == ' ')
                output = output + "0";

            else
            {
                // Calculating index for each
                // character
                int position = input.charAt(i) - 'A';
                output = output + arr[position];
            }
        }

        // Output sequence
        return output;
    }

    // Driver Function
    public static void main(String[] args)
    {
        // storing the sequence in array
        String str[] = {"2","22","222",
                        "3","33","333",
                        "4","44","444",
                        "5","55","555",
                        "6","66","666",
                        "7","77","777","7777",
                        "8","88","888",
                        "9","99","999","9999"
                    };

        String input = "GEEKSFORGEEKS";
        System.out.println(printSequence(str, input));
    }
}

// This code is contributed by Gitanjali.
```

Python3

```
 # Python3 implementation to convert
# a sentence into its equivalent
# mobile numeric keypad sequence
```

```
# Function which computes the
# sequence
def printSequence(arr, input):

# length of input string
    n = len(input)
    output = ""

    for i in range(n):

        # checking for space
        if(input[i] == ' '):
            output = output + "0"
        else:
            # calculating index for each
            # character
            position = ord(input[i]) - ord('A')
            output = output + arr[position]
    # output sequence
    return output

# Driver code
str = ["2", "22", "222",
       "3", "33", "333",
       "4", "44", "444",
       "5", "55", "555",
       "6", "66", "666",
       "7", "77", "777", "7777",
       "8", "88", "888",
       "9", "99", "999", "9999" ]

input = "GEEKSFORGEEKS";
print( printSequence(str, input))

# This code is contributed by upendra bartwal
```

C#

```
 // C# implementation to convert a
// sentence into its equivalent
// mobile numeric keypad sequence
using System;

class GFG
{

    // Function which computes the sequence
```

```
    static String printSequence(string []arr,
                            string input)
    {
        string output = "";

        // length of input string
        int n = input.Length;
        for (int i = 0; i < n; i++)
        {
            // Checking for space
            if (input[i] == ' ')
                output = output + "0";

            else
            {
                // Calculating index for each
                // character
                int position = input[i] - 'A';
                output = output + arr[position];
            }
        }

        // Output sequence
        return output;
    }

    // Driver Function
    public static void Main()
    {
        // storing the sequence in array
        string []str = {"2","22","222",
                        "3","33","333",
                        "4","44","444",
                        "5","55","555",
                        "6","66","666",
                        "7","77","777","7777",
                        "8","88","888",
                        "9","99","999","9999"
                    };

        string input = "GEEKSFORGEEKS";
        Console.WriteLine(printSequence(str, input));
    }
}

// This code is contributed by vt_m.
```

Output :

```
4333355777733366677743333557777
```

Time complexity : O(n)

Source

https://www.geeksforgeeks.org/convert-sentence-equivalent-mobile-numeric-keypad-sequence/

Chapter 32

Convert an array to reduced form | Set 1 (Simple and Hashing)

Convert an array to reduced form | Set 1 (Simple and Hashing) - GeeksforGeeks

Given an array with n distinct elements, convert the given array to a form where all elements are in range from 0 to n-1. The order of elements is same, i.e., 0 is placed in place of smallest element, 1 is placed for second smallest element, … n-1 is placed for largest element.

```
Input:  arr[] = {10, 40, 20}
Output: arr[] = {0, 2, 1}

Input:  arr[] = {5, 10, 40, 30, 20}
Output: arr[] = {0, 1, 4, 3, 2}
```

Expected time complexity is O(n Log n).

Method 1 (Simple)
A Simple Solution is to first find minimum element replace it with 0, consider remaining array and find minimum in the remaining array and replace it with 1 and so on. Time complexity of this solution is $O(n^2)$

Method 2 (Efficient)
The idea is to use hashing and sorting. Below are steps.
1) Create a temp array and copy contents of given array to temp[]. This takes O(n) time.
2) Sort temp[] in ascending order. This takes O(n Log n) time.
3) Create an empty hash table. This takes O(1) time.
4) Traverse temp[] form left to right and store mapping of numbers and their values (in converted array) in hash table. This takes O(n) time on average.

5) Traverse given array and change elements to their positions using hash table. This takes O(n) time on average.

Overall time complexity of this solution is O(n Log n).

Below are implementations of above idea.

C++

```
 // C++ program to convert an array in reduced
// form
#include <bits/stdc++.h>
using namespace std;

void convert(int arr[], int n)
{
    // Create a temp array and copy contents
    // of arr[] to temp
    int temp[n];
    memcpy(temp, arr, n*sizeof(int));

    // Sort temp array
    sort(temp, temp + n);

    // Create a hash table. Refer
    // http://tinyurl.com/zp5wgef
    unordered_map<int, int> umap;

    // One by one insert elements of sorted
    // temp[] and assign them values from 0
    // to n-1
    int val = 0;
    for (int i = 0; i < n; i++)
        umap[temp[i]] = val++;

    // Convert array by taking positions from
    // umap
    for (int i = 0; i < n; i++)
        arr[i] = umap[arr[i]];
}

void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above method
int main()
```

```
{
    int arr[] = {10, 20, 15, 12, 11, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Given Array is \n";
    printArr(arr, n);

    convert(arr , n);

    cout << "\n\nConverted Array is \n";
    printArr(arr, n);

    return 0;
}
```

Java

```
 // Java Program to convert an Array
// to reduced form
import java.util.*;

class GFG
{
    public static void convert(int arr[], int n)
    {
        // Create a temp array and copy contents
        // of arr[] to temp
        int temp[] = arr.clone();

        // Sort temp array
        Arrays.sort(temp);

        // Create a hash table.
        HashMap<Integer, Integer> umap = new HashMap<>();

        // One by one insert elements of sorted
        // temp[] and assign them values from 0
        // to n-1
        int val = 0;
        for (int i = 0; i < n; i++)
            umap.put(temp[i], val++);

        // Convert array by taking positions from
        // umap
        for (int i = 0; i < n; i++)
            arr[i] = umap.get(arr[i]);
    }
```

```
    public static void printArr(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }

    // Driver code
    public static void main(String[] args)
    {

        int arr[] = {10, 20, 15, 12, 11, 50};
        int n = arr.length;

        System.out.println("Given Array is ");
        printArr(arr, n);

        convert(arr , n);

        System.out.println("\n\nConverted Array is ");
        printArr(arr, n);

    }
}

// This code is contributed by Abhishek Panwar
```

Python3

```
 # Python3 program to convert an array
# in reduced form
def convert(arr, n):
    # Create a temp array and copy contents
    # of arr[] to temp
    temp = [arr[i] for i in range (n) ]

    # Sort temp array
    temp.sort()

    # create a map
    umap = {}


    # One by one insert elements of sorted
    # temp[] and assign them values from 0
    # to n-1
    val = 0
    for i in range (n):
        umap[temp[i]] = val
```

```
        val += 1

    # Convert array by taking positions from umap
    for i in range (n):
        arr[i] = umap[arr[i]]

def printArr(arr, n):
    for i in range(n):
        print(arr[i], end = " ")

# Driver Code
if __name__ == "__main__":
    arr = [10, 20, 15, 12, 11, 50]
    n = len(arr)
    print("Given Array is ")
    printArr(arr, n)
    convert(arr , n)
    print("\n\nConverted Array is ")
    printArr(arr, n)

# This code is contributed by Abhishek Gupta
```

Output :

```
Given Array is
10 20 15 12 11 50

Converted Array is
0 4 3 2 1 5
```

Convert an array to reduced form | Set 2 (Using vector of pairs)

This article is contributed by **Dheeraj Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : panwarabhishek345, ab_gupta

Source

https://www.geeksforgeeks.org/convert-an-array-to-reduced-form-set-1-simple-and-hashing/

Chapter 33

Convert to a string that is repetition of a substring of k length

Convert to a string that is repetition of a substring of k length - GeeksforGeeks

Given a string, find if it is possible to convert it to a string that is repetition of substring with k characters. To convert, we can replace one substring of length k with k characters.

Examples:

```
Input: str = "bdac",  k = 2
Output: True
We can either replace "bd" with "ac" or
"ac" with "bd".

Input: str = "abcbedabcabc",  k = 3
Output: True
Replace "bed" with "abc" so that the
whole string becomes repetition of "abc".

Input: str = "bcacc", k = 3
Output: False
k doesn't divide string length i.e. 5%3 != 0

Input: str = "bcacbcac", k = 2
Output: False

Input: str = "bcdbcdabcedcbcd", k = 3
Output: False
```

This can be used in compression. If we have a string where complete string is repetition except one substring, then we can use this algorithm to compress the string.

One observation is, length of string must be a multiple of k as we can replace only one substring.
The idea is declare a map **mp** which maps strings of length k to an integer denoting its count. So, if there are only two different sub-strings of length k in the map container and count of one of the sub-string is 1 then answer is true. Otherwise answer is false.

```
 // C++ program to check if a string can be converted to
// a string that has repeated substrings of length k.
#include<bits/stdc++.h>
using namespace std;

// Returns true if str can be coverted to a string
// with k repeated substrings after replacing k
// characters.
bool checkString(string str, long k)
{
    // Length of string must be a multiple of k
    int n = str.length();
    if (n%k != 0)
        return false;

    // Map to store strings of length k and their counts
    unordered_map<string, int> mp;
    for (int i=0; i<n; i+=k)
        mp[str.substr(i, k)]++;

    // If string is already a repition of k substrings,
    // return true.
    if (mp.size() == 1)
        return true;

    // If number of distinct substrings is not 2, then
    // not possible to replace a string.
    if (mp.size() != 2)
        return false;

    // One of the two distinct must appear exactly once.
    // Either the first entry appears once, or it appears
    // n/k-1 times to make other substring appear once.
    if ((mp.begin()->second == (n/k - 1)) ||
                   mp.begin()->second == 1)
       return true;

    return false;
}
```

```
// Driver code
int main()
{
    checkString("abababcd", 2)? cout << "Yes" :
                                cout << "No";
    return 0;
}
```

Output:

```
Yes
```

Source

https://www.geeksforgeeks.org/convert-string-repetition-substring-k-length/

Chapter 34

Count Substrings with equal number of 0s, 1s and 2s

Count Substrings with equal number of 0s, 1s and 2s - GeeksforGeeks

Given a string which consists of only 0s, 1s or 2s, count the number of substrings that have equal number of 0s, 1s and 2s.

Examples:

```
Input  :  str = "0102010"
Output :  2
Explanation : Substring str[2, 4] = "102" and
              substring str[4, 6] = "201" has
              equal number of 0, 1 and 2

Input : str = "102100211"
Output : 5
```

A **simple solution** is to iterate through all substring of str and checking whether they contain equal 0,1 and 2 or not. Total number of substring of str is $O(n^2)$ checking each substring for count takes O(n) times, So total time taken to solve this problem is $O(n^3)$ time with brute-force approach.

An **efficient solution** is to keep track of counts of 0, 1 and 2.

```
Let zc[i] denotes number of zeros between index 1 and i
    oc[i] denotes number of ones between index 1 and i
    tc[i] denotes number of twos between index 1 and i
for substring str[i, j] to be counted in result we should have :
    zc[i] - zc[j-1] = oc[i] - oc[j-1] = tc[i] - tc[j-1]
```

```
we can write above relation as follows :
z[i] - o[i] = z[j-1] - o[j-1]   and
z[i] - t[i] = z[j-1] - t[j-1]
```

The above relations can be tracked while looping in string, at each index i we'll calculate this difference pair and we'll check how many time this difference pair has previously occurred and we'll add that count to our result, for keeping track of previous difference pair we have used map in below code. Total time complexity of this solution is O(n log n) considering the fact that map operations, like search and insert take O(Log n) time.

```
 // C++ program to find substring with equal
// number of 0's, 1's and 2's
#include <bits/stdc++.h>
using namespace std;

// Method to count number of substring which
// has equal 0, 1 and 2
int getSubstringWithEqual012(string str)
{
    int n = str.length();

    // map to store, how many times a difference
    // pair has occurred previously
    map< pair<int, int>, int > mp;
    mp[make_pair(0, 0)] = 1;

    //  zc (Count of zeroes), oc(Count of 1s)
    //  and tc(count of twos)
    //  In starting all counts are zero
    int zc = 0, oc = 0, tc = 0;

    //  looping into string
    int res = 0;  // Initialize result
    for (int i = 0; i < n; ++i)
    {
        // incresing the count of current character
        if (str[i] == '0') zc++;
        else if (str[i] == '1') oc++;
        else tc++;  // Assuming that string doesn't contain
                    // other characters

        // making pair of differences (z[i] - o[i],
        // z[i] - t[i])
        pair<int, int> tmp = make_pair(zc - oc,
                                       zc - tc);

        // Count of previous occurrences of above pair
        // indicates that the subarrays forming from
```

```
        // every previous occurrence to this occurrence
        // is a subarray with equal number of 0's, 1's
        // and 2's
        res = res + mp[tmp];

        // increasing the count of current difference
        // pair by 1
        mp[tmp]++;
    }

    return res;
}

//  driver code to test above method
int main()
{
    string str = "0102010";
    cout << getSubstringWithEqual012(str) << endl;
    return 0;
}
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/substring-equal-number-0-1-2/

Chapter 35

Count all distinct pairs with difference equal to k

Count all distinct pairs with difference equal to k - GeeksforGeeks

Given an integer array and a positive integer k, count all distinct pairs with difference equal to k.

Examples:

```
Input: arr[] = {1, 5, 3, 4, 2}, k = 3
Output: 2
There are 2 pairs with difference 3, the pairs are {1, 4} and {5, 2}

Input: arr[] = {8, 12, 16, 4, 0, 20}, k = 4
Output: 5
There are 5 pairs with difference 4, the pairs are {0, 4}, {4, 8},
{8, 12}, {12, 16} and {16, 20}
```

Method 1 (Simple)
A simple solution is to consider all pairs one by one and check difference between every pair. Following program implements the simple solution. We run two loops: the outer loop picks the first element of pair, the inner loop looks for the other element. This solution doesn't work if there are duplicates in array as the requirement is to count only distinct pairs.

C++

```
 /* A simple program to count pairs with difference k*/
#include<iostream>
using namespace std;

int countPairsWithDiffK(int arr[], int n, int k)
```

```
{
    int count = 0;

    // Pick all elements one by one
    for (int i = 0; i < n; i++)
    {
        // See if there is a pair of this picked element
        for (int j = i+1; j < n; j++)
            if (arr[i] - arr[j] == k || arr[j] - arr[i] == k )
                  count++;
    }
    return count;
}

// Driver program to test above function
int main()
{
    int arr[] =  {1, 5, 3, 4, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << "Count of pairs with given diff is "
         << countPairsWithDiffK(arr, n, k);
    return 0;
}
```

Java

```
 // A simple Java program to
//count pairs with difference k
import java.util.*;
import java.io.*;

class GFG {

    static int countPairsWithDiffK(int arr[],
                                   int n, int k)
    {
        int count = 0;

        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {
            // See if there is a pair
            // of this picked element
            for (int j = i + 1; j < n; j++)
                if (arr[i] - arr[j] == k ||
                    arr[j] - arr[i] == k)
                    count++;
```

```
        }
        return count;
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 1, 5, 3, 4, 2 };
        int n = arr.length;
        int k = 3;
        System.out.println("Count of pairs with given diff is "
                        + countPairsWithDiffK(arr, n, k));
    }
}

// This code is contributed
// by Sahil_Bansall
```

Python3

```
 # A simple program to count pairs with difference k

def countPairsWithDiffK(arr, n, k):
    count = 0

    # Pick all elements one by one
    for i in range(0, n):

        # See if there is a pair of this picked element
        for j in range(i+1, n) :

            if arr[i] - arr[j] == k or arr[j] - arr[i] == k:
                count += 1

    return count

# Driver program
arr = [1, 5, 3, 4, 2]

n = len(arr)
k = 3
print ("Count of pairs with given diff is ",
                countPairsWithDiffK(arr, n, k))
```

C#

```
 // A simple C# program to count pairs with
```

```
// difference k
using System;

class GFG {

    static int countPairsWithDiffK(int []arr,
                                   int n, int k)
    {
        int count = 0;

        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {

            // See if there is a pair
            // of this picked element
            for (int j = i + 1; j < n; j++)
                if (arr[i] - arr[j] == k ||
                      arr[j] - arr[i] == k)
                    count++;
        }

        return count;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 1, 5, 3, 4, 2 };
        int n = arr.Length;
        int k = 3;

        Console.WriteLine("Count of pairs with "
                             + " given diff is "
               + countPairsWithDiffK(arr, n, k));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// A simple PHP program to count
// pairs with difference k

function countPairsWithDiffK($arr, $n,
                                   $k)
```

```
{
    $count = 0;

    // Pick all elements one by one
    for($i = 0; $i < $n; $i++)
    {

        // See if there is a pair of
        // this picked element
        for($j = $i + 1; $j < $n; $j++)
            if ($arr[$i] - $arr[$j] == $k or
                $arr[$j] - $arr[$i] == $k)
                $count++;
    }
    return $count;
}

    // Driver Code
    $arr = array(1, 5, 3, 4, 2);
    $n = count($arr);
    $k = 3;
    echo "Count of pairs with given diff is "
        , countPairsWithDiffK($arr, $n, $k);

// This code is contributed by anuj_67.
?>
```

Output :

```
Count of pairs with given diff is 2
```

Time Complexity of $O(n^2)$

Method 2 (Use Sorting)
We can find the count in O(nLogn) time using a O(nLogn) sorting algorithm like Merge Sort, Heap Sort, etc. Following are the detailed steps.

```
1) Initialize count as 0
2) Sort all numbers in increasing order.
3) Remove duplicates from array.
4) Do following for each element arr[i]
   a) Binary Search for arr[i] + k in subarray from i+1 to n-1.
   b) If arr[i] + k found, increment count.
5) Return count.
```

C++

```
 /* A sorting based program to count pairs with difference k*/
#include <iostream>
#include <algorithm>
using namespace std;

/* Standard binary search function */
int binarySearch(int arr[], int low, int high, int x)
{
    if (high >= low)
    {
        int mid = low + (high - low)/2;
        if (x == arr[mid])
            return mid;
        if (x > arr[mid])
            return binarySearch(arr, (mid + 1), high, x);
        else
            return binarySearch(arr, low, (mid -1), x);
    }
    return -1;
}

/* Returns count of pairs with difference k in arr[] of size n. */
int countPairsWithDiffK(int arr[], int n, int k)
{
    int count = 0, i;
    sort(arr, arr+n);  // Sort array elements

    /* code to remove duplicates from arr[] */

    // Pick a first element point
    for (i = 0; i < n-1; i++)
        if (binarySearch(arr, i+1, n-1, arr[i] + k) != -1)
            count++;

    return count;
}

// Driver program
int main()
{
    int arr[] = {1, 5, 3, 4, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << "Count of pairs with given diff is "
        << countPairsWithDiffK(arr, n, k);
    return 0;
```

```
}
```

Java

```
 // A sorting base java program to
// count pairs with difference k
import java.util.*;
import java.io.*;

class GFG {

    // Standard binary search function //
    static int binarySearch(int arr[], int low,
                                int high, int x)
    {
        if (high >= low)
        {
            int mid = low + (high - low) / 2;
            if (x == arr[mid])
                return mid;
            if (x > arr[mid])
                return binarySearch(arr, (mid + 1),
                                        high, x);
            else
                return binarySearch(arr, low,
                                    (mid - 1), x);
        }
        return -1;
    }

    // Returns count of pairs with
    // difference k in arr[] of size n.
    static int countPairsWithDiffK(int arr[], int n, int k)
    {
        int count = 0, i;

        // Sort array elements
        Arrays.sort(arr);

        // code to remove duplicates from arr[]

        // Pick a first element point
        for (i = 0; i < n - 1; i++)
            if (binarySearch(arr, i + 1, n - 1,
                             arr[i] + k) != -1)
                count++;

        return count;
```

```
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 1, 5, 3, 4, 2 };
        int n = arr.length;
        int k = 3;
        System.out.println("Count of pairs with given diff is "
                           + countPairsWithDiffK(arr, n, k));
    }
}

// This code is contributed by Sahil_Bansall
```

Python

```
 # A sorting based program to
# count pairs with difference k

# Standard binary search function
def binarySearch(arr, low, high, x):

    if (high >= low):

        mid = low + (high - low)//2
        if x == arr[mid]:
            return (mid)
        elif(x > arr[mid]):
            return binarySearch(arr, (mid + 1), high, x)
        else:
            return binarySearch(arr, low, (mid -1), x)

    return -1


# Returns count of pairs with
# difference k in arr[] of size n.
def countPairsWithDiffK(arr, n, k):

    count = 0
    arr.sort() # Sort array elements

    # code to remove
    # duplicates from arr[]

    # Pick a first element point
    for i in range (0, n - 2):
```

```
        if (binarySearch(arr, i + 1, n - 1,
                         arr[i] + k) != -1):
            count += 1


    return count

# Driver Code
arr= [1, 5, 3, 4, 2]
n = len(arr)
k = 3
print ("Count of pairs with given diff is ",
             countPairsWithDiffK(arr, n, k))

# This code is contributed
# by Shivi_Aggarwal
```

C#

```
 // A sorting base C# program to
// count pairs with difference k
using System;

class GFG {

    // Standard binary search function
    static int binarySearch(int []arr, int low,
                            int high, int x)
    {
        if (high >= low)
        {
            int mid = low + (high - low) / 2;
            if (x == arr[mid])
                return mid;
            if (x > arr[mid])
                return binarySearch(arr, (mid + 1),
                                        high, x);
            else
                return binarySearch(arr, low,
                                    (mid - 1), x);
        }

        return -1;
    }

    // Returns count of pairs with
    // difference k in arr[] of size n.
    static int countPairsWithDiffK(int []arr,
```

```
                                    int n, int k)
    {

        int count = 0, i;

        // Sort array elements
        Array.Sort(arr);

        // code to remove duplicates from arr[]

        // Pick a first element point
        for (i = 0; i < n - 1; i++)
            if (binarySearch(arr, i + 1, n - 1,
                             arr[i] + k) != -1)
                count++;

        return count;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 1, 5, 3, 4, 2 };
        int n = arr.Length;
        int k = 3;

        Console.WriteLine("Count of pairs with"
                             + " given diff is "
              + countPairsWithDiffK(arr, n, k));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// A sorting based PHP program to
// count pairs with difference k

// Standard binary search function
function binarySearch($arr, $low,
                      $high, $x)
{
    if ($high >= $low)
    {
        $mid = $low + ($high - $low)/2;
        if ($x == $arr[$mid])
```

```
            return $mid;

        if ($x > $arr[$mid])
            return binarySearch($arr, ($mid + 1),
                                $high, $x);
        else
            return binarySearch($arr, $low,
                                ($mid -1), $x);
    }
    return -1;
}

/* Returns count of pairs with
   difference k in arr[] of size n. */
function countPairsWithDiffK($arr, $n, $k)
{
    $count = 0;
    $i;

    // Sort array elements
    sort($arr);

    // Code to remove duplicates
    // from arr[]

    // Pick a first element point
    for ($i = 0; $i < $n - 1; $i++)
        if (binarySearch($arr, $i + 1, $n - 1,
                         $arr[$i] + $k) != -1)
            $count++;

    return $count;
}

    // Driver Code
    $arr = array(1, 5, 3, 4, 2);
    $n = count($arr);
    $k = 3;
    echo "Count of pairs with given diff is "
         , countPairsWithDiffK($arr, $n, $k);

// This code is contributed by anuj-67.
?>
```

Output:

```
Count of pairs with given diff is 2
```

Time complexity: The first step (sorting) takes O(nLogn) time. The second step runs binary search n times, so the time complexity of second step is also O(nLogn). Therefore, overall time complexity is O(nLogn). The second step can be optimized to O(n), see this.

Method 3 (Use Self-balancing BST)
We can also a self-balancing BST like AVL tree or Red Black tree to solve this problem. Following is detailed algorithm.

```
1) Initialize count as 0.
2) Insert all elements of arr[] in an AVL tree. While inserting,
   ignore an element if already present in AVL tree.
3) Do following for each element arr[i].
   a) Search for arr[i] + k in AVL tree, if found then increment count.
   b) Search for arr[i] - k in AVL tree, if found then increment count.
   c) Remove arr[i] from AVL tree.
```

Time complexity of above solution is also O(nLogn) as search and delete operations take O(Logn) time for a self-balancing binary search tree.

Method 4 (Use Hashing)
We can also use hashing to achieve the average time complexity as O(n) for many cases.

```
1) Initialize count as 0.
2) Insert all distinct elements of arr[] in a hash map.  While inserting,
   ignore an element if already present in the hash map.
3) Do following for each element arr[i].
   a) Look for arr[i] + k in the hash map, if found then increment count.
   b) Look for arr[i] - k in the hash map, if found then increment count.
   c) Remove arr[i] from hash table.
```

A very simple case where hashing works in O(n) time is the case where range of values is very small. For example, in the following implementation, range of numbers is assumed to be 0 to 99999. A simple hashing technique to use values as index can be used.

```
 /* An efficient program to count pairs with difference k when the range
   numbers is small */
#define MAX 100000
int countPairsWithDiffK(int arr[], int n, int k)
{
    int count = 0;  // Initialize count

    // Initialize empty hashmap.
    bool hashmap[MAX] = {false};

    // Insert array elements to hashmap
    for (int i = 0; i < n; i++)
        hashmap[arr[i]] = true;
```

```
    for (int i = 0; i < n; i++)
    {
        int x = arr[i];
        if (x - k >= 0 && hashmap[x - k])
            count++;
        if (x + k < MAX && hashmap[x + k])
            count++;
        hashmap[x] = false;
    }
    return count;
}
```

Method 5 (Use Sorting)

Sort the array arr

Take two pointers, l and r, both pointing to 1st element

Take the difference arr[r] – arr[l]

- If value diff is K, increment count and move both pointers to next element
- if value diff > k, move l to next element
- if value diff < k, move r to next element

return count

C++

```
 /* A sorting based program to count pairs with difference k*/
#include <iostream>
#include <algorithm>
using namespace std;

/* Returns count of pairs with difference k in arr[] of size n. */
int countPairsWithDiffK(int arr[], int n, int k)
{
    int count = 0;
    sort(arr, arr+n);  // Sort array elements

    int l = 0;
    int r = 0;
    while(r < n)
    {
         if(arr[r] - arr[l] == k)
        {
              count++;
              l++;
              r++;
        }
```

```
         else if(arr[r] - arr[l] > k)
              l++;
         else // arr[r] - arr[l] < sum
              r++;
    }
    return count;
}

// Driver program to test above function
int main()
{
    int arr[] =  {1, 5, 3, 4, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << "Count of pairs with given diff is "
         << countPairsWithDiffK(arr, n, k);
    return 0;
}
```

Java

```
 // A sorting based Java program to
// count pairs with difference k
import java.util.*;

class GFG {

/* Returns count of pairs with
difference k in arr[] of size n. */
static int countPairsWithDiffK(int arr[], int n,
                                           int k)
{
    int count = 0;
    Arrays.sort(arr); // Sort array elements

    int l = 0;
    int r = 0;
    while(r < n)
    {
        if(arr[r] - arr[l] == k)
        {
            count++;
            l++;
            r++;
        }
        else if(arr[r] - arr[l] > k)
            l++;
        else // arr[r] - arr[l] < sum
```

```
            r++;
    }
    return count;
}

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = {1, 5, 3, 4, 2};
    int n = arr.length;
    int k = 3;
    System.out.println("Count of pairs with given diff is " +
                        countPairsWithDiffK(arr, n, k));
}
}

// This code is contributed by Prerna Saini
```

C#

```
 // A sorting based C# program to count
// pairs with difference k
using System;

class GFG {

    /* Returns count of pairs with
    difference k in arr[] of size n. */
    static int countPairsWithDiffK(int []arr,
                                int n, int k)
    {
        int count = 0;

        // Sort array elements
        Array.Sort(arr);

        int l = 0;
        int r = 0;
        while(r < n)
        {
            if(arr[r] - arr[l] == k)
            {
                count++;
                l++;
                r++;
            }
            else if(arr[r] - arr[l] > k)
                l++;
```

```
            else // arr[r] - arr[l] < sum
                r++;
        }
        return count;
    }

    // Driver program to test above function
    public static void Main()
    {
        int []arr = {1, 5, 3, 4, 2};
        int n = arr.Length;
        int k = 3;
        Console.Write("Count of pairs with "
                        + "given diff is " +
            countPairsWithDiffK(arr, n, k));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A sorting based program to count
// pairs with difference k

// Returns count of pairs with
// difference k in arr[] of size n.
function countPairsWithDiffK( $arr, $n, $k)
{
    $count = 0;

    // Sort array elements
    sort($arr);

    $l = 0;
    $r = 0;
    while($r < $n)
    {
        if($arr[$r] - $arr[$l] == $k)
        {
            $count++;
            $l++;
            $r++;
        }
        else if($arr[$r] - $arr[$l] > $k)
            $l++;
```

```
        // arr[r] - arr[l] < sum
        else
            $r++;
    }
    return $count;
}

    // Driver Code
    $arr = array(1, 5, 3, 4, 2);
    $n =count($arr);
    $k = 3;
    echo "Count of pairs with given diff is "
        , countPairsWithDiffK($arr, $n, $k);

// This code is contributed by anuj_67,
?>
```

Output:

```
Count of pairs with given diff is 2
```

Time Complexity: O(nlogn)

Improved By : nitin mittal, vt_m, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/count-pairs-difference-equal-k/

Chapter 36

Count all pairs with given XOR

Count all pairs with given XOR - GeeksforGeeks

Given an array of distinct positive integers and a number x, find the number of pairs of integers in the array whose XOR is equal to x.
Examples:

```
Input : arr[] = {5, 4, 10, 15, 7, 6}, x = 5
Output : 1
Explanation :  (10 ^ 15) = 5

Input : arr[] = {3, 6, 8, 10, 15, 50}, x = 5
Output : 2
Explanation : (3 ^ 6) = 5 and (10 ^ 15) = 5
```

A **Simple solution** is to traverse each element and check if there's another number whose XOR with it is equal to x. This solution takes $O(n^2)$ time.

An **efficient solution** of this problem take O(n) time. The idea is based on the fact that arr[i] ^ arr[j] is equal to x if and only if arr[i] ^ x is equal to arr[j].

```
1) Initialize result as 0.
2) Create an empty hash set "s".
3) Do following for each element arr[i] in arr[]
   (a)    If x ^ arr[i] is in "s", then increment result by 1.
   (b)    Insert arr[i] into the hash set "s".
3) return result.
```

C++

```
 // C++ program to Count all pair with given XOR
// value x
#include<bits/stdc++.h>

using namespace std;

// Returns count of pairs in arr[0..n-1] with XOR
// value equals to x.
int xorPairCount(int arr[], int n, int x)
{
    int result = 0; // Initialize result

    // create empty set that stores the visiting
    // element of array.
    // Refer below post for details of unordered_set
    // https://www.geeksforgeeks.org/unorderd_set-stl-uses/
    unordered_set<int> s;

    for (int i=0; i<n ; i++)
    {
        // If there exist an element in set s
        // with XOR equals to x^arr[i], that means
        // there exist an element such that the
        // XOR of element with arr[i] is equal to
        // x, then increment count.
        if (s.find(x^arr[i]) != s.end())
            result++;

        // Make element visited
        s.insert(arr[i]);
    }

    // return total count of pairs with XOR equal to x
    return result;
}

// driver program
int main()
{
    int arr[] = {5 , 4 ,10, 15, 7, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 5;
    cout << "Count of pairs with given XOR = "
         << xorPairCount(arr, n, x);
    return 0;
}
```

Python3

```
 #Python3 program to count all the pair with given xor

#Returns count of pairs in arr[0..n-1] with XOR
#value equals to x.
def xorPairCount(arr,n,x):
    result = 0 #Initialize result
    #create empty set that stores the visiting
    #element of array.
    s = set()
    for i in range(0,n):
        #If there exist an element in set s
        #with XOR equals to x^arr[i], that means
        #there exist an element such that the
        #XOR of element with arr[i] is equal to
        #x, then increment count.
        if(x^arr[i] in s):
            result = result +1
        #Make element visited
        s.add(arr[i])
    return result
# Driver Code

if __name__ == "__main__":
    arr = [5, 4, 10, 15, 7, 6]
    n = len(arr)
    x = 5
    print("Count of pair with given XOR = "
         +str(xorPairCount(arr,n,x)))

#program by Anubhav Natani
```

Output:

```
Count of pairs with given XOR = 1
```

Time complexity : O(n)

How to handle duplicates?
The above efficient solution doesn't work if there are duplicates in input array. For example, the above solution produces different results for {2, 2, 5} and {5, 2, 2}. To handle duplicates, we store counts of occurrences of all elements. We use unordered_map instead of unordered_set.

C++

```
 // C++ program to Count all pair with given XOR
```

```
// value x
#include<bits/stdc++.h>

using namespace std;

// Returns count of pairs in arr[0..n-1] with XOR
// value equals to x.
int xorPairCount(int arr[], int n, int x)
{
    int result = 0; // Initialize result

    // create empty map that stores counts of
    // individual elements of array.
    unordered_map<int, int> m;

    for (int i=0; i<n ; i++)
    {
        int curr_xor =  x^arr[i];

        // If there exist an element in map m
        // with XOR equals to x^arr[i], that means
        // there exist an element such that the
        // XOR of element with arr[i] is equal to
        // x, then increment count.
        if (m.find(curr_xor) != m.end())
            result += m[curr_xor];

        // Increment count of current element
        m[arr[i]]++;
    }


    // return total count of pairs with XOR equal to x
    return result;
}

// driver program
int main()
{
    int arr[] = {2, 5, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 0;
    cout << "Count of pairs with given XOR = "
         << xorPairCount(arr, n, x);
    return 0;
}
```

Output:

```
Count of pairs with given XOR = 1
```

Time complexity : O(n)

Improved By : NerdCaps, AnubhavNatani

Source

https://www.geeksforgeeks.org/count-pairs-given-xor/

Chapter 37

Count distinct elements in every window of size k

Count distinct elements in every window of size k - GeeksforGeeks

Given an array of size n and an integer k, return the of count of distinct numbers in all windows of size k.

Example:

```
Input:  arr[] = {1, 2, 1, 3, 4, 2, 3};
            k = 4
Output:
3
4
4
3

Explanation:
First window is {1, 2, 1, 3}, count of distinct numbers is 3
Second window is {2, 1, 3, 4} count of distinct numbers is 4
Third window is {1, 3, 4, 2} count of distinct numbers is 4
Fourth window is {3, 4, 2, 3} count of distinct numbers is 3
```

Source: Microsoft Interview Question

A **Simple Solution** is to traverse the given array, consider every window in it and count distinct elements in the window. Below is implementation of simple solution.

C++

```
 // Simple C++ program to count distinct elements in every
// window of size k
```

```
#include <iostream>
using namespace std;

// Counts distinct elements in window of size k
int countWindowDistinct(int win[], int k)
{
    int dist_count = 0;

    // Traverse the window
    for (int i=0; i<k; i++)
    {
        // Check if element arr[i] exists in arr[0..i-1]
        int j;
        for (j=0; j<i; j++)
           if (win[i] == win[j])
              break;
        if (j==i)
            dist_count++;
    }
    return dist_count;
}

// Counts distinct elements in all windows of size k
void countDistinct(int arr[], int n, int k)
{
    // Traverse through every window
    for (int i=0; i<=n-k; i++)
       cout << countWindowDistinct(arr+i, k) << endl;
}

// Driver program
int main()
{
    int arr[] = {1, 2, 1, 3, 4, 2, 3},  k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    countDistinct(arr, n, k);
    return 0;
}
```

Java

```
 // Simple Java program to count distinct elements in every
// window of size k

import java.util.Arrays;

class Test
{
```

```
    // Counts distinct elements in window of size k
    static int countWindowDistinct(int win[], int k)
    {
        int dist_count = 0;

        // Traverse the window
        for (int i = 0; i < k; i++)
        {
            // Check if element arr[i] exists in arr[0..i-1]
            int j;
            for (j = 0; j < i; j++)
               if (win[i] == win[j])
                  break;
            if (j == i)
                dist_count++;
        }
        return dist_count;
    }

    // Counts distinct elements in all windows of size k
    static void countDistinct(int arr[], int n, int k)
    {
        // Traverse through every window
        for (int i = 0; i <= n - k; i++)
           System.out.println(countWindowDistinct
                              (Arrays.copyOfRange(arr, i, arr.length), k));
    }

    // Driver method
    public static void main(String args[])
    {
         int arr[] = {1, 2, 1, 3, 4, 2, 3},  k = 4;

         countDistinct(arr, arr.length, k);
    }
}
```

Output:

```
3
4
4
3
```

Time complexity of the above solution is $O(nk^2)$. We can improve time complexity to O(nkLok) by modifying countWindowDistinct() to use sorting. The function can further be optimized to use hashing to find distinct elements in a window. With hashing the time complexity becomes O(nk). Below is a different approach that works in O(n) time.

An **Efficient Solution** is to use the count of the previous window while sliding the window. The idea is to create a hash map that stores elements of the current window. When we slide the window, we remove an element from the hash and add an element. We also keep track of distinct elements. Below is the algorithm.

1) Create an empty hash map. Let hash map be hM

2) Initialize distinct element count 'dist_count' as 0.

3) Traverse through the first window and insert elements of the first window to hM. The elements are used as key and their counts as the value in hM. Also, keep updating 'dist_count'

4) Print 'dist_count' for the first window.

3) Traverse through the remaining array (or other windows).
….a) Remove the first element of the previous window.
…….If the removed element appeared only once
……..remove it from hM and do "dist_count–"
…….Else (appeared multiple times in hM)
……..decrement its count in hM

….a) Add the current element (last element of the new window)
…….If the added element is not present in hM
……..add it to hM and do "dist_count++"
…….Else (the added element appeared multiple times)
……..increment its count in hM

Below is an implementation of above approach.

Java

```
 // An efficient Java program to count distinct elements in
// every window of size k
import java.util.HashMap;

class CountDistinctWindow
{
    static void countDistinct(int arr[], int k)
    {
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM =
                       new HashMap<Integer, Integer>();

        // initialize distinct element  count for
        // current window
        int dist_count = 0;

        // Traverse the first window and store count
        // of every element in hash map
        for (int i = 0; i < k; i++)
        {
            if (hM.get(arr[i]) == null)
```

```
    {
        hM.put(arr[i], 1);
        dist_count++;
    }
    else
    {
       int count = hM.get(arr[i]);
       hM.put(arr[i], count+1);
    }
}

// Print count of first window
System.out.println(dist_count);

// Traverse through the remaining array
for (int i = k; i < arr.length; i++)
{

    // Remove first element of previous window
    // If there was only one occurrence, then
    // reduce distinct count.
    if (hM.get(arr[i-k]) == 1)
    {
        hM.remove(arr[i-k]);
        dist_count--;
    }
    else // reduce count of the removed element
    {
       int count = hM.get(arr[i-k]);
       hM.put(arr[i-k], count-1);
    }

    // Add new element of current window
    // If this element appears first time,
    // increment distinct element count
    if (hM.get(arr[i]) == null)
    {
        hM.put(arr[i], 1);
        dist_count++;
    }
    else // Increment distinct element count
    {
       int count = hM.get(arr[i]);
       hM.put(arr[i], count+1);
    }

   // Print count of current window
    System.out.println(dist_count);
```

```
        }
    }

    // Driver method
    public static void main(String arg[])
    {
        int arr[] =  {1, 2, 1, 3, 4, 2, 3};
        int k = 4;
        countDistinct(arr, k);
    }
}
```

C++

```
 #include <iostream>
#include <map>
using namespace std;

void countDistinct(int arr[], int k, int n)
{
    // Creates an empty hashmap hm
    map<int, int> hm;

    // initialize distinct element count for current window
    int dist_count = 0;

    // Traverse the first window and store count
    // of every element in hash map
    for (int i = 0; i < k; i++)
    {
       if (hm[arr[i]] == 0)
       {
           dist_count++;
       }
    hm[arr[i]] += 1;
    }

   // Print count of first window
  cout << dist_count << endl;

   // Traverse through the remaining array
   for (int i = k; i < n; i++)
   {
     // Remove first element of previous window
     // If there was only one occurrence, then reduce distinct count.
     if (hm[arr[i-k]] == 1)
     {
        dist_count--;
```

```
     }
   // reduce count of the removed element
     hm[arr[i-k]] -= 1;

   // Add new element of current window
   // If this element appears first time,
   // increment distinct element count

    if (hm[arr[i]] == 0)
    {
       dist_count++;
    }
    hm[arr[i]] += 1;

  // Print count of current window
    cout << dist_count << endl;
  }
}

int main()
{
   int arr[] = {1, 2, 1, 3, 4, 2, 3};
   int size = sizeof(arr)/sizeof(arr[0]);
   int k = 4;
   countDistinct(arr, k, size);

   return 0;
}
//This solution is contributed by Aditya Goel
```

Output:

```
3
4
4
3
```

Time complexity of the above solution is O(n).

This article is contributed by **Piyush**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/count-distinct-elements-in-every-window-of-size-k/

Chapter 38

Count divisible pairs in an array

Count divisible pairs in an array - GeeksforGeeks

Given an array, count pairs in the array such that one element of pair divides other.

Examples:

```
Input  : arr[] = {1, 2, 3}
Output : 2
The two pairs are (1, 2) and (1, 3)

Input : arr[] = {2, 3, 5, 7}
Output: 0
```

C++

```
 // CPP program to count divisible pairs.
#include <iostream>
using namespace std;

int countDivisibles(int arr[], int n)
{
    int res = 0;

    // Iterate through all pairs
    for (int i=0; i<n; i++)
      for (int j=i+1; j<n; j++)

         // Increment count if one divides
         // other
         if (arr[i] % arr[j] == 0 ||
             arr[j] % arr[i] == 0)
```

```
            res++;

    return res;
}

int main()
{
    int a[] = {1, 2, 3, 9};
    int n = sizeof(a) / sizeof(a[0]);
    cout << countDivisibles(a, n);
    return 0;
}
```

Java

```
 // Java program to count
// divisible pairs.

class GFG {

// Function returns count
// of divisible pairs
static int countDivisibles(int arr[],
                           int n)
{
    int res = 0;

    // Iterate through all pairs
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)

        // Increment count if
        // one divides other
        if (arr[i] % arr[j] == 0 ||
            arr[j] % arr[i] == 0)
            res++;

    return res;
}

// Driver Code
public static void main(String[] args)
{
    int a[] = new int[]{1, 2, 3, 9};
    int n = a.length;
    System.out.print(countDivisibles(a, n));
}
}
```

```
// This code is contributed by Smitha.
```

C#

```
 // Java program to count
// divisible pairs.
using System;

class GFG {

// Function returns count
// of divisible pairs
static int countDivisibles(int []arr,
                           int n)
{
    int res = 0;

    // Iterate through all pairs
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)

        // Increment count if
        // one divides other
        if (arr[i] % arr[j] == 0 ||
            arr[j] % arr[i] == 0)
            res++;

    return res;
}

// Driver Code
public static void Main(String[] args)
{
    int[] a = new int[4] {1, 2, 3, 9};
    int n = a.Length;
    Console.Write(countDivisibles(a, n));
}
}

// This code is contributed by Smitha.
```

PHP

```
 <?php
// PHP program to count divisible pairs.
function countDivisibles($arr, $n)
```

```
{
    $res = 0;

    // Iterate through all pairs
    for ($i = 0; $i < $n; $i++)
    for ($j = $i + 1; $j < $n; $j++)

        // Increment count if one divides
        // other
        if ($arr[$i] % $arr[$j] == 0 ||
            $arr[$j] % $arr[$i] == 0)
            $res++;

    return $res;
}
$a = array(1, 2, 3, 9);
$n = count($a);
echo (countDivisibles($a, $n));
?>
```

Output:

```
4
```

Efficient solution for small ranged numbers.
1) Insert all elements of array in a hash table.
2) Find maximum element in the array.
3) For every array element, search multiples of it (till maximum) in the hash table. If found, increment result.
Different cases like negative numbers and repetitions can also be handled here with slight modifications to the approach.

Improved By : Smitha Dinesh Semwal, manishshaw1

Source

https://www.geeksforgeeks.org/count-divisible-pairs-array/

Chapter 39

Count elements that are divisible by at-least one element in another array

Count elements that are divisible by at-least one element in another array - GeeksforGeeks

Given two arrays arr1[] and arr2[]. The task is to find the count of such elements in the first array whose at-least one factor is present in the second array.

Examples:

```
Input : arr1[] = {10, 2, 13, 4, 15} ; arr2[] = {2, 4, 5, 6}
Output : 4
There is no factor of 13 which is present in the
second array. Except 13, factors of the rest 4
elements of the first array is present in the
second array.

Input : arr1[] = {11, 13, 17, 15} ; arr2[] = {3, 7, 9, 5}
Output : 1
```

The idea is to insert all elements of the second array into a hash such that the lookup for factors can be done in constant time. Now, traverse the first array and for every element generate all of the factors starting from 1 and check if any of the factors is present in the hash or not.

Below is the implementation of the above approach:

```
 // CPP program to find count of
// elements in first arary whose
// atleast one factor is present
```

```
// in second array.
#include <bits/stdc++.h>
using namespace std;

// Util function to count the elements
// in first array whose atleast
// one factor is present in second array
int elementCount(int arr1[], int n1, int arr2[], int n2)
{

    // counter to count number of elements
    int count = 0;

    // Hash of second array elements
    unordered_set<int> hash;
    for (int i = 0; i < n2; i++)
        hash.insert(arr2[i]);

    // loop to traverse through array elements
    // and check for its factors
    for (int i = 0; i < n1; i++) {

        // generate factors of elements
        // of first array
        for (int j = 1; j * j <= arr1[i]; j++) {

            // Check if j is a factor
            if (arr1[i] % j == 0) {

                // check if the factor is present in
                // second array using the hash
                if ((hash.find(j) != hash.end()) ||
                        (hash.find(arr1[i] / j) != hash.end())) {
                    count++;
                    break;
                }
            }
        }
    }

    return count;
}

// Driver code
int main()
{
    int arr1[] = { 10, 2, 13, 4, 15 };
    int arr2[] = { 2, 4, 5, 6 };
```

```
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    cout << elementCount(arr1, n1, arr2, n2);

    return 0;
}
```

Output:

```
4
```

Source

https://www.geeksforgeeks.org/count-elements-that-are-divisible-by-at-least-one-element-in-another-array/

Chapter 40

Count items common to both the lists but with different prices

Count items common to both the lists but with different prices - GeeksforGeeks

Given two lists **list1** and **list2** containing **m** and **n** items respectively. Each item is associated with two fields: name and price. The problem is to count items which are in both the lists but with different prices.

Examples:

```
Input : list1[] = {{"apple", 60}, {"bread", 20},
                   {"wheat", 50}, {"oil", 30}}
    list2[] = {{"milk", 20}, {"bread", 15},
                   {"wheat", 40}, {"apple", 60}}
Output : 2
bread and wheat are the two items common to both the
lists but with different prices.
```

Source: Cognizant Interview Experience | Set 5.

Method 1 (Naive Approach): Using two nested loops compare each item of **list1** with all the items of **list2**. If match is found with a different price then increment the **count**.

CPP

```
 // C++ implementation to count items common to both
// the lists but with different prices
#include <bits/stdc++.h>
```

```
using namespace std;

// details of an item
struct item
{
    string name;
    int price;
};

// function to count items common to both
// the lists but with different prices
int countItems(item list1[], int m,
               item list2[], int n)
{
    int count = 0;

    // for each item of 'list1' check if it is in 'list2'
    // but with a different price
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)

            if ((list1[i].name.compare(list2[j].name) == 0) &&
                (list1[i].price != list2[j].price))
                count++;

    // required count of items
    return count;
}

// Driver program to test above
int main()
{
    item list1[] = {{"apple", 60}, {"bread", 20},
                    {"wheat", 50}, {"oil", 30}};
    item list2[] = {{"milk", 20}, {"bread", 15},
                   {"wheat", 40}, {"apple", 60}};

    int m = sizeof(list1) / sizeof(list1[0]);
    int n = sizeof(list2) / sizeof(list2[0]);

    cout << "Count = "
         << countItems(list1, m, list2, n);

    return 0;
}
```

Python3

```
 # Python implementation to
# count items common to both
# the lists but with different
# prices

# function to count items
# common to both
# the lists but with different prices
def countItems(list1, list2):
    count = 0

    # for each item of 'list1'
    # check if it is in 'list2'
    # but with a different price
    for i in list1:
        for j in list2:

            if i[0] == j[0] and i[1] != j[1]:
                count += 1

    # required count of items
    return count

# Driver program to test above
list1 = [("apple", 60), ("bread", 20),
            ("wheat", 50), ("oil", 30)]
list2 = [("milk", 20), ("bread", 15),
            ("wheat", 40), ("apple", 60)]

print("Count = ", countItems(list1, list2))

# This code is contributed by Ansu Kumari.
```

Output:

```
Count = 2
```

Time Complexity: O(m*n).
Auxiliary Space: O(1).

Method 2 (Binary Search): Sort the **list2** in alphabetical order of its items name. Now, for each item of **list1** check whether it in present in **list2** using binary search technique and get its price from **list2**. If prices are different the increment the **count**.

```
 // C++ implementation to count items common to both
// the lists but with different prices
#include <bits/stdc++.h>
```

```
using namespace std;

// details of an item
struct item
{
    string name;
    int price;
};

// comparator function used for sorting
bool compare(struct item a, struct item b)
{
    return (a.name.compare(b.name) <= 0);
}

// function to search 'str' in 'list2[]'. If it exists then
// price associated with 'str' in 'list2[]' is being
// returned else -1 is returned. Here binary serach
// trechnique is being applied for searching
int binary_search(item list2[], int low, int high, string str)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        // if true the item 'str' is in 'list2'
        if (list2[mid].name.compare(str) == 0)
            return list2[mid].price;

        else if (list2[mid].name.compare(str) < 0)
            low = mid + 1;

        else
            high = mid - 1;
    }

    // item 'str' is not in 'list2'
    return -1;
}

// function to count items common to both
// the lists but with different prices
int countItems(item list1[], int m,
               item list2[], int n)
{
    // sort 'list2' in alphabetcal order of
    // items name
```

```
    sort(list2, list2 + n, compare);

    // initial count
    int count = 0;

    for (int i = 0; i < m; i++) {

        // get the price of item 'list1[i]' from 'list2'
        // if item in not present in second list then
        // -1 is being obtained
        int r = binary_search(list2, 0, n-1, list1[i].name);

        // if item is present in list2 with a
        // different price
        if ((r != -1) && (r != list1[i].price))
            count++;
    }

    // required count of items
    return count;
}

// Driver program to test above
int main()
{
    item list1[] = {{"apple", 60}, {"bread", 20},
                    {"wheat", 50}, {"oil", 30}};
    item list2[] = {{"milk", 20}, {"bread", 15},
                   {"wheat", 40}, {"apple", 60}};

    int m = sizeof(list1) / sizeof(list1[0]);
    int n = sizeof(list2) / sizeof(list2[0]);

    cout << "Count = "
         << countItems(list1, m, list2, n);

    return 0;
}
```

Output:

```
Count = 2
```

Time Complexity: ($m*\log_2 n$).
Auxiliary Space: O(1).
For efficiency, the list with maximum number of elements should sorted and used for binary search.

Method 3 (Efficient Approach): Create a hash table with **(key, value)** tuple as **(item name, price)**. Insert the elements of **list1** in the hash table. Now, for each element of **list2** check if it is the hash table or not. If it is present, then check if its price is different then the value from the hash table. If so then increment the **count**.

```
 // C++ implementation to count items common to both
// the lists but with different prices
#include <bits/stdc++.h>

using namespace std;

// details of an item
struct item
{
    string name;
    int price;
};

// function to count items common to both
// the lists but with different prices
int countItems(item list1[], int m,
               item list2[], int n)
{
    // 'um' implemented as hash table that contains
    // item name as the key and price as the value
    // associated with the key
    unordered_map<string, int> um;
    int count = 0;

    // insert elements of 'list1' in 'um'
    for (int i = 0; i < m; i++)
        um[list1[i].name] = list1[i].price;

    // for each element of 'list2' check if it is
    // present in 'um' with a different price
    // value
    for (int i = 0; i < n; i++)
        if ((um.find(list2[i].name) != um.end()) &&
            (um[list2[i].name] != list2[i].price))
            count++;

    // required count of items
    return count;
}

// Driver program to test above
int main()
{
```

```
    item list1[] = {{"apple", 60}, {"bread", 20},
                    {"wheat", 50}, {"oil", 30}};
    item list2[] = {{"milk", 20}, {"bread", 15},
                    {"wheat", 40}, {"apple", 60}};

    int m = sizeof(list1) / sizeof(list1[0]);
    int n = sizeof(list2) / sizeof(list2[0]);

    cout << "Count = "
         << countItems(list1, m, list2, n);

    return 0;
}
```

Output:

```
Count = 2
```

Time Complexity: O(m + n).
Auxiliary Space: O(m).
For efficiency, the list having minimum number of elements should be inserted in the hash table.

Source

https://www.geeksforgeeks.org/count-items-common-lists-different-prices/

Chapter 41

Count maximum points on same line

Count maximum points on same line - GeeksforGeeks

Given N point on a 2D plane as pair of (x, y) co-ordinates, we need to find maximum number of point which lie on the same line.

Examples:

```
Input : points[] = {-1, 1}, {0, 0}, {1, 1},
                   {2, 2}, {3, 3}, {3, 4}
Output : 4
Then maximum number of point which lie on same
line are 4, those point are {0, 0}, {1, 1}, {2, 2},
{3, 3}
```

We can solve above problem by following approach – For each point p, calculate its slope with other points and use a map to record how many points have same slope, by which we can find out how many points are on same line with p as their one point. For each point keep doing the same thing and update the maximum number of point count found so far.

Some things to note in implementation are:
1) if two point are (x1, y1) and (x2, y2) then their slope will be (y2 – y1) / (x2 – x1) which can be a double value and can cause precision problems. To get rid of the precision problems, we treat slope as pair ((y2 – y1), (x2 – x1)) instead of ratio and reduce pair by their gcd before inserting into map. In below code points which are vertical or repeated are treated separately.

2) If we use unordered_map in c++ or HashMap in Java for storing the slope pair, then total time complexity of solution will be O(n^2)

```
/* C/C++ program to find maximum number of point
```

```
which lie on same line */
#include <bits/stdc++.h>
#include <boost/functional/hash.hpp>

using namespace std;

// method to find maximum colinear point
int maxPointOnSameLine(vector< pair<int, int> > points)
{
    int N = points.size();
    if (N < 2)
        return N;

    int maxPoint = 0;
    int curMax, overlapPoints, verticalPoints;

    // here since we are using unordered_map
    // which is based on hash function
    //But by default we don't have hash function for pairs
    //so we'll use hash function defined in Boost library
    unordered_map<pair<int, int>, int,boost::
              hash<pair<int, int> > > slopeMap;

    // looping for each point
    for (int i = 0; i < N; i++)
    {
        curMax = overlapPoints = verticalPoints = 0;

        // looping from i + 1 to ignore same pair again
        for (int j = i + 1; j < N; j++)
        {
            // If both point are equal then just
            // increase overlapPoint count
            if (points[i] == points[j])
                overlapPoints++;

            // If x co-ordinate is same, then both
            // point are vertical to each other
            else if (points[i].first == points[j].first)
                verticalPoints++;

            else
            {
                int yDif = points[j].second - points[i].second;
                int xDif = points[j].first - points[i].first;
                int g = __gcd(xDif, yDif);

                // reducing the difference by their gcd
```

```
                yDif /= g;
                xDif /= g;

                // increasing the frequency of current slope
                // in map
                slopeMap[make_pair(yDif, xDif)]++;
                curMax = max(curMax, slopeMap[make_pair(yDif, xDif)]);
            }

            curMax = max(curMax, verticalPoints);
        }

        // updating global maximum by current point's maximum
        maxPoint = max(maxPoint, curMax + overlapPoints + 1);

        // printf("maximum colinear point
        // which contains current point
        // are : %d\n", curMax + overlapPoints + 1);
        slopeMap.clear();
    }

    return maxPoint;
}

// Driver code
int main()
{
    const int N = 6;
    int arr[N][2] = {{-1, 1}, {0, 0}, {1, 1}, {2, 2},
                     {3, 3}, {3, 4}};

    vector< pair<int, int> > points;
    for (int i = 0; i < N; i++)
        points.push_back(make_pair(arr[i][0], arr[i][1]));

    cout << maxPointOnSameLine(points) << endl;

    return 0;
}
```

Output:

```
4
```

Improved By : PortgasDAce

Source

https://www.geeksforgeeks.org/count-maximum-points-on-same-line/

Chapter 42

Count number of Distinct Substring in a String

Count number of Distinct Substring in a String - GeeksforGeeks

Given a string, count all distinct substrings of the given string.

Examples

```
Input : abcd
Output : abcd abc ab a bcd bc b cd c d
All Elements are Distinct

Input : aaa
Output : aaa aa a aa a a
All elements are not Distinct
```

Prerequisite : Print subarrays of a given array

The idea is to use hash table (HashSet in Java) to store all generated substrings. Finally we return size of the HashSet.

```
 // Java program to count all distinct substrings in a string
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class DistinctSubstring {

    public static int distinctSubstring(String str)
    {
        // Put all distinct substring in a HashSet
        Set<String> result = new HashSet<String>();
```

```
        // List All Substrings
        for (int i = 0; i <= str.length(); i++) {
            for (int j = i + 1; j <= str.length(); j++) {

                // Add each substring in Set
                result.add(str.substring(i, j));
            }
        }

        // Return size of the HashSet
        return result.size();
    }

    // Driver Code
    public static void main(String[] args)
    {
        String str = "aaaa";
        System.out.println(distinctSubstring(str));
    }
}
```

Output:

```
4
```

How to print the distinct substrings?

```
 // Java program to count all distinct substrings in a string
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class DistinctSubstring {

    public static Set<String> distinctSubstring(String str)
    {

        // Put all distinct substring in a HashSet
        Set<String> result = new HashSet<String>();

        // List All Substrings
        for (int i = 0; i <= str.length(); i++) {
            for (int j = i + 1; j <= str.length(); j++) {

                // Add each substring in Set
```

```
                result.add(str.substring(i, j));
            }
        }

        // Return the HashSet
        return result;
    }

    // Driver Code
    public static void main(String[] args)
    {
        String str = "aaaa";
        Set<String> subs = distinctSubstring(str);

        System.out.println("Distinct Substrings are: ");
        for (String s : subs) {
            System.out.println(s);
        }
    }
}
```

Output:

```
Distinct Substrings are:
aa
aaa
a
aaaa
```

Source

https://www.geeksforgeeks.org/count-number-of-distinct-substring-in-a-string/

Chapter 43

Count number of elements between two given elements in array

Count number of elements between two given elements in array - GeeksforGeeks

Given an unsorted array of n elements and also given two points num1 and num2. The task is to count number of elements occurs between the given points (excluding num1 and num2).

If there are multiple occurrences of num1 and num2, we need to consider leftmost occurrence of num1 and rightmost occurrence of num2.

Examples:

```
Input : arr[] = {3 5 7 6 4 9 12 4 8}
        num1 = 5
        num2 = 4
Output : 5
Number of elements between leftmost occurrence
of 5 and rightmost occurrence of 4 is five.

Input : arr[] = {4, 6, 8, 3, 6, 2, 8, 9, 4}
        num1 = 4
        num2 = 4
Output : 7

Input : arr[] = {4, 6, 8, 3, 6, 2, 8, 9, 4}
        num1 = 4
        num2 = 10
Output : 0
```

The solution should traverse array only once in all cases (when single or both elements are not present)

The idea is to traverse array from left and find first occurrence of num1. If we reach end, we return 0. Then we traverse from rightmost element and find num2. We traverse only till the point which is greater than index of num1. If we reach end, we return 0. If we found both elements, we return count using indexes of found elements.

CPP

```
 // Program to count number of elements between
// two given elements.
#include <bits/stdc++.h>
using namespace std;

// Function to count number of elements
// occurs between the elements.
int getCount(int arr[], int n, int num1, int num2)
{
    // Find num1
    int i = 0;
    for (i = 0; i < n; i++)
        if (arr[i] == num1)
            break;

    // If num1 is not present or present at end
    if (i >= n-1)
        return 0;

    // Find num2
    int j;
    for (j = n-1; j >= i+1; j--)
        if (arr[j] == num2)
            break;

    // If num2 is not present
    if (j == i)
        return 0;

    // return number of elements between
    // the two elements.
    return (j - i - 1);
}

// Driver Code
int main()
{
    int arr[] = { 3, 5, 7, 6, 4, 9, 12, 4, 8 };
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    int num1 = 5, num2 = 4;
    cout << getCount(arr, n, num1, num2);
    return 0;
}
```

Java

```
 // Program to count number of elements
// between two given elements.
import java.io.*;

class GFG
{
    // Function to count number of elements
    // occurs between the elements.
    static int getCount(int arr[], int n,
                            int num1, int num2)
    {
        // Find num1
        int i = 0;
        for (i = 0; i < n; i++)
            if (arr[i] == num1)
                break;

        // If num1 is not present
        // or present at end
        if (i >= n - 1)
            return 0;

        // Find num2
        int j;
        for (j = n - 1; j >= i + 1; j--)
            if (arr[j] == num2)
                break;

        // If num2 is not present
        if (j == i)
            return 0;

        // return number of elements
        // between the two elements.
        return (j - i - 1);
    }

    // Drvier program
    public static void main (String[] args)
    {
```

```
        int arr[] = { 3, 5, 7, 6, 4, 9, 12, 4, 8 };
        int n = arr.length;
        int num1 = 5, num2 = 4;
        System.out.println( getCount(arr, n, num1, num2));

    }
}
// This code is contributed by vt_m
```

Python3

```
 # Python Program to count number of elements between
# two given elements.

# Function to count number of elements
# occurs between the elements.
def getCount(arr, n, num1, num2):

    # Find num1
    for i in range(0,n):
        if (arr[i] == num1):
            break

    #If num1 is not present or present at end
    if (i >= n-1):
        return 0

    # Find num2
    for j in range(n-1, i+1, -1):
        if (arr[j] == num2):
            break

    # If num2 is not present
    if (j == i):
        return 0

    # return number of elements between
    # the two elements.
    return (j - i - 1)

# Driver Code
arr= [ 3, 5, 7, 6, 4, 9, 12, 4, 8 ]
n=len(arr)
num1 = 5
num2 = 4
print(getCount(arr, n, num1, num2))

# This code is contributed by SHARIQ_JMI
```

C#

```
 // C# Program to count number of elements
// between two given elements.
using System;

class GFG  {

    // Function to count number of elements
    // occurs between the elements.
    static int getCount(int []arr, int n,
                        int num1, int num2)
    {

        // Find num1
        int i = 0;
        for (i = 0; i < n; i++)
            if (arr[i] == num1)
                break;

        // If num1 is not present
        // or present at end
        if (i >= n - 1)
            return 0;

        // Find num2
        int j;
        for (j = n - 1; j >= i + 1; j--)
            if (arr[j] == num2)
                break;

        // If num2 is not present
        if (j == i)
            return 0;

        // return number of elements
        // between the two elements.
        return (j - i - 1);
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {3, 5, 7, 6, 4, 9, 12, 4, 8};
        int n = arr.Length;
        int num1 = 5, num2 = 4;
        Console.WriteLine(getCount(arr, n, num1, num2));
```

```
    }
}

// This article is contributed by vt_m.
```

PHP

```
 <?php
// Program to count number
// of elements between
// two given elements.

// Function to count
// number of elements
// occurs between the
// elements.
function getCount($arr, $n,
                  $num1, $num2)
{

    // Find num1
    $i = 0;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] == $num1)
            break;

    // If num1 is not present
    // or present at end
    if ($i >= $n - 1)
        return 0;

    // Find num2
    $j;
    for ($j = $n - 1; $j >= $i + 1; $j--)
        if ($arr[$j] == $num2)
            break;

    // If num2 is not present
    if ($j == $i)
        return 0;

    // return number of elements
    // betweenthe two elements.
    return ($j - $i - 1);
}

// Driver Code
$arr = array(3, 5, 7, 6, 4, 9, 12, 4, 8);
```

```
$n = sizeof($arr);
$num1 = 5; $num2 = 4;
echo(getCount($arr, $n, $num1, $num2));

// This code is contributed by Ajit.
?>
```

Output:

```
5
```

How to handle multiple queries?
For handling multiple queries, we can use hashing and store leftmost and rightmost indexes for every element present in array. Once we have stored these, we can answer all queries in O(1) time.

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/count-number-elements-two-given-elements-array/

Chapter 44

Count number of triplets with product equal to given number

Count number of triplets with product equal to given number - GeeksforGeeks

Given an array of distinct integers(considering only positive numbers) and a number 'm', find the number of triplets with product equal to 'm'.

Examples:

```
Input : arr[] = { 1, 4, 6, 2, 3, 8}
            m = 24
Output : 3
{1, 4, 6} {1, 3, 8} {4, 2, 3}

Input : arr[] = { 0, 4, 6, 2, 3, 8}
            m = 18
Output : 0
```

Asked in : Microsoft

A **Naive approach** is to consider each and every triplet one by one and count if their product is equal to m.

C/C++

```
 // C++ program to count triplets with given
// product m
#include <iostream>
using namespace std;

// Function to count such triplets
```

```
int countTriplets(int arr[], int n, int m)
{
    int count = 0;

    // Consider all triplets and count if
    // their product is equal to m
    for (int i = 0; i < n - 2; i++)
        for (int j = i + 1; j < n - 1; j++)
            for (int k = j + 1; k < n; k++)
                if (arr[i] * arr[j] * arr[k] == m)
                    count++;

    return count;
}

// Drivers code
int main()
{
    int arr[] = { 1, 4, 6, 2, 3, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 24;

    cout << countTriplets(arr, n, m);

    return 0;
}
```

Java

```
 // Java program to count triplets with given
// product m

class GFG {
    // Method to count such triplets
    static int countTriplets(int arr[], int n, int m)
    {
        int count = 0;

        // Consider all triplets and count if
        // their product is equal to m
        for (int i = 0; i < n - 2; i++)
            for (int j = i + 1; j < n - 1; j++)
                for (int k = j + 1; k < n; k++)
                    if (arr[i] * arr[j] * arr[k] == m)
                        count++;

        return count;
    }
```

```
    // Driver method
    public static void main(String[] args)
    {
        int arr[] = { 1, 4, 6, 2, 3, 8 };
        int m = 24;

        System.out.println(countTriplets(arr, arr.length, m));
    }
}
```

C#

```
 // C# program to count triplets
// with given product m
using System;

public class GFG {

    // Method to count such triplets
    static int countTriplets(int[] arr, int n, int m)
    {
        int count = 0;

        // Consider all triplets and count if
        // their product is equal to m
        for (int i = 0; i < n - 2; i++)
            for (int j = i + 1; j < n - 1; j++)
                for (int k = j + 1; k < n; k++)
                    if (arr[i] * arr[j] * arr[k] == m)
                        count++;

        return count;
    }

    // Driver method
    public static void Main()
    {
        int[] arr = { 1, 4, 6, 2, 3, 8 };
        int m = 24;

        Console.WriteLine(countTriplets(arr, arr.Length, m));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to count triplets
// with given product m

// Function to count such triplets
function countTriplets($arr, $n, $m)
{
    $count = 0;

    // Consider all triplets and count if
    // their product is equal to m
    for ( $i = 0; $i < $n - 2; $i++)
        for ( $j = $i + 1; $j < $n - 1; $j++)
            for ($k = $j + 1; $k < $n; $k++)
                if ($arr[$i] * $arr[$j] * $arr[$k] == $m)
                    $count++;

    return $count;
}

    // Driver code
    $arr = array(1, 4, 6, 2, 3, 8);
    $n = sizeof($arr);
    $m = 24;
    echo countTriplets($arr, $n, $m);

// This code is contributed by jit_t.
?>
```

Output:

```
3
```

Time Complexity: O(n^3)

An **Efficient Method** is to use Hashing.

1. Store all the elements in a hash_map with their index.
2. Consider all pairs(i, j) and check following:
    - If (arr[i]*arr[j] !=0 && (m % arr[i]*arr[j]) == 0), If yes, then search for (m / (arr[i]*arr[j]) in the map.
    - Also check m / (arr[i]*arr[j]) is not equal to arr[i] and arr[j].
    - Also check that current triplet is not counted previously by using index stored in the map.
    - If all the above conditions are satisfied, then increment count.

3. Return count.

C++

```
 // C++ program to count triplets with given
// product m
#include <bits/stdc++.h>
using namespace std;

// Function to count such triplets
int countTriplets(int arr[], int n, int m)
{
    // Store all the elements in a set
    unordered_map<int, int> occ;
    for (int i = 0; i < n; i++)
        occ[arr[i]] = i;

    int count = 0;

    // Consider all pairs and check for a
    // third number so their product is equal to m
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            // Check if current pair divides m or not
            // If yes, then search for (m / arr[i]*arr[j])
            if ((arr[i] * arr[j] <= m) && (arr[i] * arr[j] != 0) && (m % (arr[i] * arr[j]) == 0)
                int check = m / (arr[i] * arr[j]);
                auto it = occ.find(check);

                // Check if the third number is present
                // in the map and it is not equal to any
                // other two elements and also check if
                // this triplet is not counted already
                // using their indexes
                if (check != arr[i] && check != arr[j]
                    && it != occ.end() && it->second > i
                    && it->second > j)
                    count++;
            }
        }
    }

    // Return number of triplets
    return count;
}

// Drivers code
int main()
```

```
{
    int arr[] = { 1, 4, 6, 2, 3, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 24;

    cout << countTriplets(arr, n, m);

    return 0;
}
```

Java

```
 // Java program to count triplets with given
// product m

import java.util.HashMap;

class GFG {
    // Method to count such triplets
    static int countTriplets(int arr[], int n, int m)
    {
        // Store all the elements in a set
        HashMap<Integer, Integer> occ = new HashMap<Integer, Integer>(n);
        for (int i = 0; i < n; i++)
            occ.put(arr[i], i);

        int count = 0;

        // Consider all pairs and check for a
        // third number so their product is equal to m
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                // Check if current pair divides m or not
                // If yes, then search for (m / arr[i]*arr[j])
                if ((arr[i] * arr[j] <= m) && (arr[i] * arr[j] != 0) && (m % (arr[i] * arr[j]) ==
                    int check = m / (arr[i] * arr[j]);

                    occ.containsKey(check);

                    // Check if the third number is present
                    // in the map and it is not equal to any
                    // other two elements and also check if
                    // this triplet is not counted already
                    // using their indexes
                    if (check != arr[i] && check != arr[j]
                        && occ.containsKey(check) && occ.get(check) > i
                        && occ.get(check) > j)
                        count++;
```

```
                }
            }
        }

        // Return number of triplets
        return count;
    }

    // Driver method
    public static void main(String[] args)
    {
        int arr[] = { 1, 4, 6, 2, 3, 8 };
        int m = 24;

        System.out.println(countTriplets(arr, arr.length, m));
    }
}
```

Output:

```
3
```

Time Complexity : $O(n^2)$
Auxiliary Space : O(n)

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/count-number-triplets-product-equal-given-number/

Chapter 45

Count of index pairs with equal elements in an array

Count of index pairs with equal elements in an array - GeeksforGeeks

Given an array of **n** elements. The task is to count the total number of indices (i, j) such that arr[i] = arr[j] and i != j

Examples :

```
Input : arr[] = {1, 1, 2}
Output : 1
As arr[0] = arr[1], the pair of indices is (0, 1)

Input : arr[] = {1, 1, 1}
Output : 3
As arr[0] = arr[1], the pair of indices is (0, 1),
(0, 2) and (1, 2)

Input : arr[] = {1, 2, 3}
Output : 0
```

Method 1 (Brute Force):
For each index i, find element after it with same value as arr[i]. Below is the implementation of this approach:

C++

```
 // C++ program to count of pairs with equal
// elements in an array.
#include<bits/stdc++.h>
```

```
using namespace std;

// Return the number of pairs with equal
// values.
int countPairs(int arr[], int n)
{
    int ans = 0;

    // for each index i and j
    for (int i = 0; i < n; i++)
        for (int j = i+1; j < n; j++)

            // finding the index with same
            // value but different index.
            if (arr[i] == arr[j])
                ans++;
    return ans;
}

// Driven Program
int main()
{
    int arr[] = { 1, 1, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to count of pairs with equal
// elements in an array.
class GFG {

    // Return the number of pairs with equal
    // values.
    static int countPairs(int arr[], int n)
    {
        int ans = 0;

        // for each index i and j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)

                // finding the index with same
                // value but different index.
                if (arr[i] == arr[j])
                    ans++;
```

```
        return ans;
    }

    //driver code
    public static void main (String[] args)
    {
        int arr[] = { 1, 1, 2 };
        int n = arr.length;

        System.out.println(countPairs(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to
# count of pairs with equal
# elements in an array.

# Return the number of
# pairs with equal values.
def countPairs(arr, n):

    ans = 0

    # for each index i and j
    for i in range(0 , n):
        for j in range(i + 1, n):

            # finding the index
            # with same value but
            # different index.
            if (arr[i] == arr[j]):
                ans += 1
    return ans

# Driven Code
arr = [1, 1, 2 ]
n = len(arr)
print(countPairs(arr, n))

# This code is contributed
# by Smitha
```

C#

```
 // C# program to count of pairs with equal
// elements in an array.
using System;

class GFG {

    // Return the number of pairs with equal
    // values.
    static int countPairs(int []arr, int n)
    {
        int ans = 0;

        // for each index i and j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)

                // finding the index with same
                // value but different index.
                if (arr[i] == arr[j])
                    ans++;
        return ans;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = { 1, 1, 2 };
        int n = arr.Length;

        Console.WriteLine(countPairs(arr, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to count of
// pairs with equal elements
// in an array.

// Return the number of pairs
// with equal values.
function countPairs( $arr, $n)
{
    $ans = 0;
```

```
    // for each index i and j
    for ( $i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)

            // finding the index with same
            // value but different index.
            if ($arr[$i] == $arr[$j])
                $ans++;
    return $ans;
}

// Driven Code
$arr = array( 1, 1, 2 );
$n = count($arr);
echo countPairs($arr, $n) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
1
```

Time Complexity : $O(n^2)$

Method 2 (Efficient approach):
The idea is to count the frequency of each number and then find the number of pairs with equal elements. Suppose, a number x appears k times at index i_1, i_2,….,i_k. Then pick any two indexes i_x and i_y which will be counted as 1 pair. Similarly, i_y and i_x can also be pair. So, choose ${}^nC_2$ is the number of pairs such that arr[i] = arr[j] = x.

Below is the implementation of this approach:

C++

```
 // C++ program to count of index pairs with
// equal elements in an array.
#include<bits/stdc++.h>
using namespace std;

// Return the number of pairs with equal
// values.
int countPairs(int arr[], int n)
{
    unordered_map<int, int> mp;
```

```
    // Finding frequency of each number.
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Calculating pairs of each value.
    int ans = 0;
    for (auto it=mp.begin(); it!=mp.end(); it++)
    {
        int count = it->second;
        ans += (count * (count - 1))/2;
    }

    return ans;
}

// Driven Program
int main()
{
    int arr[] = {1, 1, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Output :

```
1
```

Time Complexity : O(n)

Source:
http://stackoverflow.com/questions/26772364/efficient-algorithm-for-counting-number-of-pairs-of-identical-elemen
comment42124861_26772516

Improved By : ash_maurya, vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/count-index-pairs-equal-elements-array/

Chapter 46

Count of strings that can be formed from another string using each character at-most once

Count of strings that can be formed from another string using each character at-most once - GeeksforGeeks

Given two strings str1 and str2, the task is to print the number of times *str2* can be formed using characters of *str1*. However, a character at any index of str1 can only be used once in the formation of str2.

Examples:

> **Input:** str1 = “arajjhupoot”, str2 = “rajput”
> **Output:** 1
> str2 can only be formed once using characters of str1.
>
> **Input:** str1 = “foreeksgekseg”, str2 = “geeks”
> **Output:** 2

Approach: Since the problem has a restriction on using characters of str1 only once to form str2. If one character has been used to form one str2, it cannot be used in forming another str2. Every character of str2 must be present in str1 at least for the formation of one str1. If all the characters of str2 are already present in str1, then the character which has the minimum occurrence in str1 will be the number of str2's that can be formed using the characters of str1 once. Below are the steps:

- Create an hash-array which stores the number of occurrences of each character of str.
- Iterate for all the characters of str2, and find the minimum most occurrences of every character in str1.

- Return the minimum occurence which will be the answer.

Below is the implementation of the above approach:

```
 /// C++ program to print the number of times
// str2 can be formed from str1 using the
// characters of str1 only once
#include <bits/stdc++.h>
using namespace std;

// Function to find the number of str2
// that can be formed using characters of str1
int findNumberOfTimes(string str1, string str2)
{
    int freq[26] = { 0 };

    int l1 = str1.length();

    // iterate and mark the frequencies of
    // all characters in str1
    for (int i = 0; i < l1; i++)
        freq[str1[i] - 'a'] += 1;

    int l2 = str2.length();
    int count = INT_MAX;

    // find the minimum frequency of
    // every character in str1
    for (int i = 0; i < l2; i++)
        count = min(count, freq[str2[i] - 'a']);

    return count;
}

// Driver Code
int main()
{
    string str1 = "foreeksgekseg";
    string str2 = "geeks";

    cout << findNumberOfTimes(str1, str2)
         << endl;

    return 0;
}
```

Output:

2

Time Complexity: O(max(l1,l2)), where l1 and l2 are length of str1 and str2 respectively.

Source

https://www.geeksforgeeks.org/count-of-strings-that-can-be-formed-from-another-string-using-each-character-at-m

Chapter 47

Count of substrings of a binary string containing K ones

Count of substrings of a binary string containing K ones - GeeksforGeeks

Given a binary string of length N and an integer K, we need to find out how many substrings of this string are exist which contains exactly K ones.

Examples:

```
Input : s = "10010"
        K = 1
Output : 9
The 9 substrings containing one 1 are,
"1", "10", "100", "001", "01", "1",
"10", "0010" and "010"
```

In this problem we need to find count of substrings which contains exactly K ones or in other words sum of digits in those substring is K. We first create a prefix sum array and loop over that and stop when sum value is greater than or equal to K. Now if sum at current index is (K + a) then we know that substring sum, from all those indices where sum is (a), till current index will be K, so count of indices having sum (a), will be added to result. This procedure is explained with an example below,

```
string s = "100101"
K = 2
prefix sum array = [1, 1, 1, 2, 2, 3]

So, at index 3,    we have prefix sum 2,
Now total indices from where sum is 2, is 1
```

```
so result = 1

Substring considered = ["1001"]
At index 4,        we have prefix sum 2,
Now total indices from where sum is 2, is
1 so result = 2

Substring considered = ["1001", "10010"]
At index 5,     we have prefix sum 3,
Now total indices from where sum is 2,
is 3 so result = 5
Substring considered = ["1001", "10010",
"00101", "0101", "101"]
```

So we need to track two things, prefix sum and frequency of particular sum. In below code, instead of storing complete prefix sum, only prefix sum at current index is stored using one variable and frequency of sums in stored in an array. Total time complexity of solution is O(N).

C++

```
 // C++ program to find count of substring containing
// exactly K ones
#include <bits/stdc++.h>
using namespace std;

// method returns total number of substring having K ones
int countOfSubstringWithKOnes(string s, int K)
{
    int N = s.length();
    int res = 0;
    int countOfOne = 0;
    int freq[N + 1] = {0};

    // initialize index having zero sum as 1
    freq[0] = 1;

    // loop over binary characters of string
    for (int i = 0; i < N; i++) {

        // update countOfOne variable with value
        // of ith character
        countOfOne += (s[i] - '0');

        // if value reaches more than K, then
        // update result
        if (countOfOne >= K) {
```

```
            // add frequency of indices, having
            // sum (current sum - K), to the result
            res += freq[countOfOne - K];
        }

        // update freqency of one's count
        freq[countOfOne]++;
    }
    return res;
}

// Driver code to test above methods
int main()
{
    string s = "10010";
    int K = 1;
    cout << countOfSubstringWithKOnes(s, K) << endl;
    return 0;
}
```

Java

```
 // Java program to find count of substring
// containing exactly K ones
import java.io.*;

public class GFG {

    // method returns total number of
    // substring having K ones
    static int countOfSubstringWithKOnes(
                            String s, int K)
    {
        int N = s.length();
        int res = 0;
        int countOfOne = 0;
        int []freq = new int[N+1];

        // initialize index having zero
        // sum as 1
        freq[0] = 1;

        // loop over binary characters
        // of string
        for (int i = 0; i < N; i++) {

            // update countOfOne variable
```

```
            // with value of ith character
            countOfOne += (s.charAt(i) - '0');

            // if value reaches more than
            // K, then update result
            if (countOfOne >= K) {

                // add frequency of indices,
                // having sum (current sum - K),
                // to the result
                res += freq[countOfOne - K];
            }

            // update freqency of one's count
            freq[countOfOne]++;
        }

        return res;
    }

    // Driver code to test above methods
    static public void main (String[] args)
    {
        String s = "10010";
        int K = 1;

        System.out.println(
            countOfSubstringWithKOnes(s, K));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# program to find count of substring
// containing exactly K ones
using System;

public class GFG {

    // method returns total number of
    // substring having K ones
    static int countOfSubstringWithKOnes(
                            string s, int K)
    {
        int N = s.Length;
        int res = 0;
```

```
        int countOfOne = 0;
        int []freq = new int[N+1];

        // initialize index having zero
        // sum as 1
        freq[0] = 1;

        // loop over binary characters
        // of string
        for (int i = 0; i < N; i++) {

            // update countOfOne variable
            // with value of ith character
            countOfOne += (s[i] - '0');

            // if value reaches more than
            // K, then update result
            if (countOfOne >= K) {

                // add frequency of indices,
                // having sum (current sum - K),
                // to the result
                res += freq[countOfOne - K];
            }

            // update freqency of one's count
            freq[countOfOne]++;
        }

        return res;
    }

    // Driver code to test above methods
    static public void Main ()
    {
        string s = "10010";
        int K = 1;

        Console.WriteLine(
            countOfSubstringWithKOnes(s, K));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
```

```
// PHP program to find count
// of substring containing
// exactly K ones

// method returns total number
// of substring having K ones
function countOfSubstringWithKOnes($s, $K)
{
    $N = strlen($s);
    $res = 0;
    $countOfOne = 0;
    $freq = array();
    for ($i = 0; $i <= $N; $i++)
        $freq[$i] = 0;

    // initialize index
    // having zero sum as 1
    $freq[0] = 1;


    // loop over binary
    // characters of string
    for ($i = 0; $i < $N; $i++)
    {

        // update countOfOne
        // variable with value
        // of ith character
        $countOfOne += ($s[$i] - '0');

        // if value reaches more
        // than K, then update result
        if ($countOfOne >= $K)
        {

            // add frequency of indices,
            // having sum (current sum - K),
            // to the result
            $res = $res + $freq[$countOfOne - $K];
        }

        // update freqency
        // of one's count
        $freq[$countOfOne]++;
    }
    return $res;
}
```

```
// Driver code
$s = "10010";
$K = 1;
echo countOfSubstringWithKOnes($s, $K) ,"\n";

// This code is contributed by m_kit
?>
```

Output:

```
9
```

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/count-substrings-binary-string-containing-k-ones/

Chapter 48

Count pairs from two linked lists whose sum is equal to a given value

Count pairs from two linked lists whose sum is equal to a given value - GeeksforGeeks

Given two linked lists(can be sorted or unsorted) of size **n1** and **n2** of distinct elements. Given a value **x**. The problem is to count all pairs from both lists whose sum is equal to the given value **x**.

Note: The pair has an element from each linked list.

Examples:

```
Input : list1 = 3->1->5->7
        list2 = 8->2->5->3
        x = 10
Output : 2
The pairs are:
(5, 5) and (7, 3)

Input : list1 = 4->3->5->7->11->2->1
        list2 = 2->3->4->5->6->8-12
        x = 9
Output : 5
```

Method 1 (Naive Approach): Using two loops pick elements from both the linked lists and check whether the sum of the pair is equal to **x** or not.

C/C++

```
 // C++ implementation to count pairs from both linked
// lists  whose sum is equal to a given value
#include <bits/stdc++.h>
using namespace std;

/* A Linked list node */
struct Node
{
  int data;
  struct Node* next;
};

// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
  /* allocate node */
  struct Node* new_node =
          (struct Node*) malloc(sizeof(struct Node));

  /* put in the data  */
  new_node->data  = new_data;

  /* link the old list to the new node */
  new_node->next = (*head_ref);

  /* move the head to point to the new node */
  (*head_ref)    = new_node;
}

// function to count all pairs from both the linked lists
// whose sum is equal to a given value
int countPairs(struct Node* head1, struct Node* head2, int x)
{
    int count = 0;

    struct Node *p1, *p2;

    // traverse the 1st linked list
    for (p1 = head1; p1 != NULL; p1 = p1->next)

        // for each node of 1st list
        // traverse the 2nd list

        for (p2 = head2; p2 != NULL; p2 = p2->next)

            // if sum of pair is equal to 'x'
            // increment count
```

```
            if ((p1->data + p2->data) == x)
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    // create linked list1 3->1->5->7
    push(&head1, 7);
    push(&head1, 5);
    push(&head1, 1);
    push(&head1, 3);

    // create linked list2 8->2->5->3
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 2);
    push(&head2, 8);

    int x = 10;

    cout << "Count = "
         << countPairs(head1, head2, x);
    return 0;
}
```

Java

```
 // Java implementation to count pairs from both linked
// lists  whose sum is equal to a given value

// Note : here we use java.util.LinkedList for
// linked list implementation

import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;

class GFG
{
    // method to count all pairs from both the linked lists
    // whose sum is equal to a given value
```

```
    static int countPairs(LinkedList<Integer> head1, LinkedList<Integer> head2, int x)
    {
        int count = 0;

        // traverse the 1st linked list
        Iterator<Integer> itr1 = head1.iterator();
        while(itr1.hasNext())
        {
            // for each node of 1st list
            // traverse the 2nd list
            Iterator<Integer> itr2 = head2.iterator();

            while(itr2.hasNext())
            {
                // if sum of pair is equal to 'x'
                // increment count
                if ((itr1.next() + itr2.next()) == x)
                    count++;
            }
        }

        // required count of pairs
        return count;
    }

    // Driver method
    public static void main(String[] args)
    {
        Integer arr1[] = {3, 1, 5, 7};
        Integer arr2[] = {8, 2, 5, 3};

        // create linked list1 3->1->5->7
        LinkedList<Integer> head1 = new LinkedList<>(Arrays.asList(arr1));

        // create linked list2 8->2->5->3
        LinkedList<Integer> head2 = new LinkedList<>(Arrays.asList(arr2));

        int x = 10;

        System.out.println("Count = " + countPairs(head1, head2, x));
    }
}
```

Output:

```
Count = 2
```

Time Complexity: O(n1*n2)
Auxiliary Space: O(1)

Method 2 (Sorting): Sort the 1st linked list in ascending order and the 2nd linked list in descending order using merge sort technique. Now traverse both the lists from left to right in the following way:

Algorithm:

```
countPairs(list1, list2, x)
  Initialize count = 0
  while list != NULL and list2 != NULL
     if (list1->data + list2->data) == x
        list1 = list1->next
        list2 = list2->next
        count++
    else if (list1->data + list2->data) > x
        list2 = list2->next
    else
        list1 = list1->next

  return count
```

For simplicity, the implementation given below assumes that list1 is sorted in ascending order and list2 is sorted in descending order.

C/C++

```
 // C++ implementation to count pairs from both linked
// lists  whose sum is equal to a given value
#include <bits/stdc++.h>
using namespace std;

/* A Linked list node */
struct Node
{
  int data;
  struct Node* next;
};

// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
  /* allocate node */
  struct Node* new_node =
```

```
          (struct Node*) malloc(sizeof(struct Node));

  /* put in the data  */
  new_node->data  = new_data;

  /* link the old list to the new node */
  new_node->next = (*head_ref);

  /* move the head to point to the new node */
  (*head_ref)    = new_node;
}

// function to count all pairs from both the linked
// lists whose sum is equal to a given value
int countPairs(struct Node* head1, struct Node* head2,
                                                int x)
{
    int count = 0;

    // sort head1 in ascending order and
    // head2 in descending order
    // sort (head1), sort (head2)
    // For simplicity both lists are considered to be
    // sorted in the respective orders

    // traverse both the lists from left to right
    while (head1 != NULL && head2 != NULL)
    {
        // if this sum is equal to 'x', then move both
        // the lists to next nodes and increment 'count'
        if ((head1->data + head2->data) == x)
        {
            head1 = head1->next;
            head2 = head2->next;
            count++;
        }

        // if this sum is greater than x, then
        // move head2 to next node
        else if ((head1->data + head2->data) > x)
            head2 = head2->next;

        // else move head1 to next node
        else
            head1 = head1->next;
    }

    // required count of pairs
```

```
    return count;
}

// Driver program to test above
int main()
{
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    // create linked list1 1->3->5->7
    // assumed to be in ascending order
    push(&head1, 7);
    push(&head1, 5);
    push(&head1, 3);
    push(&head1, 1);

    // create linked list2 8->5->3->2
    // assumed to be in descending order
    push(&head2, 2);
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 8);

    int x = 10;

    cout << "Count = "
         << countPairs(head1, head2, x);
    return 0;
}
```

Java

```
 // Java implementation to count pairs from both linked
// lists  whose sum is equal to a given value

// Note : here we use java.util.LinkedList for
// linked list implementation

import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;

class GFG
{
    // method to count all pairs from both the linked lists
    // whose sum is equal to a given value
    static int countPairs(LinkedList<Integer> head1, LinkedList<Integer> head2, int x)
```

```
{
    int count = 0;

    // sort head1 in ascending order and
    // head2 in descending order
    Collections.sort(head1);
    Collections.sort(head2,Collections.reverseOrder());

    // traverse both the lists from left to right
    Iterator<Integer> itr1 = head1.iterator();
    Iterator<Integer> itr2 = head2.iterator();

    Integer num1 = itr1.hasNext() ? itr1.next() : null;
    Integer num2 = itr2.hasNext() ? itr2.next() : null;

    while(num1 != null && num2 != null)
    {

        // if this sum is equal to 'x', then move both
        // the lists to next nodes and increment 'count'

        if ((num1 + num2) == x)
        {
            num1 = itr1.hasNext() ? itr1.next() : null;
            num2 = itr2.hasNext() ? itr2.next() : null;

            count++;
        }

        // if this sum is greater than x, then
        // move itr2 to next node
        else if ((num1 + num2) > x)
            num2 = itr2.hasNext() ? itr2.next() : null;

        // else move itr1 to next node
        else
            num1 = itr1.hasNext() ? itr1.next() : null;

    }

    // required count of pairs
    return count;
}

// Driver method
public static void main(String[] args)
{
    Integer arr1[] = {3, 1, 5, 7};
```

```
        Integer arr2[] = {8, 2, 5, 3};

        // create linked list1 3->1->5->7
        LinkedList<Integer> head1 = new LinkedList<>(Arrays.asList(arr1));

        // create linked list2 8->2->5->3
        LinkedList<Integer> head2 = new LinkedList<>(Arrays.asList(arr2));

        int x = 10;

        System.out.println("Count = " + countPairs(head1, head2, x));
    }
}
```

Output:

```
Count = 2
```

Time Complexity: O(n1*logn1) + O(n2*logn2)
Auxiliary Space: O(1)
Sorting will change the order of nodes. If order is important, then copy of the linked lists can be created and used.

Method 3 (Hashing): Hash table is implemented using unordered_set in C++. We store all first linked list elements in hash table. For elements of second linked list, we subtract every element from **x** and check the result in hash table. If result is present, we increment the **count**.

C++

```
 // C++ implementation to count pairs from both linked
// lists whose sum is equal to a given value
#include <bits/stdc++.h>
using namespace std;

/* A Linked list node */
struct Node
{
  int data;
  struct Node* next;
};

// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
  /* allocate node */
```

```
  struct Node* new_node =
          (struct Node*) malloc(sizeof(struct Node));

  /* put in the data  */
  new_node->data  = new_data;

  /* link the old list to the new node */
  new_node->next = (*head_ref);

  /* move the head to point to the new node */
  (*head_ref)    = new_node;
}

// function to count all pairs from both the linked
// lists whose sum is equal to a given value
int countPairs(struct Node* head1, struct Node* head2,
                                                int x)
{
    int count = 0;

    unordered_set<int> us;

    // insert all the elements of 1st list
    // in the hash table(unordered_set 'us')
    while (head1 != NULL)
    {
        us.insert(head1->data);

        // move to next node
        head1 = head1->next;
    }

    // for each element of 2nd list
    while (head2 != NULL)
    {
        // find (x - head2->data) in 'us'
        if (us.find(x - head2->data) != us.end())
            count++;

        // move to next node
        head2 = head2->next;
    }
    // required count of pairs
    return count;
}

// Driver program to test above
int main()
```

```
{
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    // create linked list1 3->1->5->7
    push(&head1, 7);
    push(&head1, 5);
    push(&head1, 1);
    push(&head1, 3);

    // create linked list2 8->2->5->3
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 2);
    push(&head2, 8);

    int x = 10;

    cout << "Count = "
         << countPairs(head1, head2, x);
    return 0;
}
```

Java

```
 // Java implementation to count pairs from both linked
// lists  whose sum is equal to a given value

// Note : here we use java.util.LinkedList for
// linked list implementation

import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;

class GFG
{
    // method to count all pairs from both the linked lists
    // whose sum is equal to a given value
    static int countPairs(LinkedList<Integer> head1, LinkedList<Integer> head2, int x)
    {
        int count = 0;

        HashSet<Integer> us = new HashSet<Integer>();

        // insert all the elements of 1st list
        // in the hash table(unordered_set 'us')
```

```
        Iterator<Integer> itr1 = head1.iterator();
        while (itr1.hasNext())
        {
            us.add(itr1.next());

        }

        Iterator<Integer> itr2 = head2.iterator();
        // for each element of 2nd list
        while (itr2.hasNext())
        {
            // find (x - head2->data) in 'us'
            if (us.add(x - itr2.next()))
                count++;

        }

        // required count of pairs
        return count;
    }

    // Driver method
    public static void main(String[] args)
    {
        Integer arr1[] = {3, 1, 5, 7};
        Integer arr2[] = {8, 2, 5, 3};

        // create linked list1 3->1->5->7
        LinkedList<Integer> head1 = new LinkedList<>(Arrays.asList(arr1));

        // create linked list2 8->2->5->3
        LinkedList<Integer> head2 = new LinkedList<>(Arrays.asList(arr2));

        int x = 10;

        System.out.println("Count = " + countPairs(head1, head2, x));
    }
}
```

Output:

```
Count = 2
```

Time Complexity: O(n1 + n2)
Auxiliary Space: O(n1), hash table should be created of the array having smaller size so as to reduce the space complexity.

Source

https://www.geeksforgeeks.org/count-pairs-two-linked-lists-whose-sum-equal-given-value/

Chapter 49

Count pairs from two sorted arrays whose sum is equal to a given value x

Count pairs from two sorted arrays whose sum is equal to a given value x - GeeksforGeeks

Given two sorted arrays of size **m** and **n** of distinct elements. Given a value **x**. The problem is to count all pairs from both arrays whose sum is equal to **x**.
Note: The pair has an element from each array.

Examples :

```
Input : arr1[] = {1, 3, 5, 7}
        arr2[] = {2, 3, 5, 8}
        x = 10

Output : 2
The pairs are:
(5, 5) and (7, 3)

Input : arr1[] = {1, 2, 3, 4, 5, 7, 11}
        arr2[] = {2, 3, 4, 5, 6, 8, 12}
        x = 9

Output : 5
```

Method 1 (Naive Approach): Using two loops pick elements from both the arrays and check whether the sum of the pair is equal to **x** or not.

C++

```
 // C++ implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
#include <bits/stdc++.h>
using namespace std;

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;

    // generating pairs from
    // both the arrays
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)

            // if sum of pair is equal
            // to 'x' increment count
            if ((arr1[i] + arr2[j]) == x)
                count++;

    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    int x = 10;
    cout << "Count = "
         << countPairs(arr1, arr2, m, n, x);
    return 0;
}
```

Java

```
 // Java implementation to count pairs from
// both sorted arrays whose sum is equal
// to a given value
```

```
import java.io.*;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1,
             int []arr2, int m, int n, int x)
    {
        int count = 0;

        // generating pairs from
        // both the arrays
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)

                // if sum of pair is equal
                // to 'x' increment count
                if ((arr1[i] + arr2[j]) == x)
                    count++;

        // required count of pairs
        return count;
    }

    // Driver Code

    public static void main (String[] args)
    {
        int arr1[] = {1, 3, 5, 7};
        int arr2[] = {2, 3, 5, 8};
        int m = arr1.length;
        int n = arr2.length;
        int x = 10;

        System.out.println( "Count = "
        + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

Python3

```
 # python implementation to count
# pairs from both sorted arrays
```

```
# whose sum is equal to a given
# value

# function to count all pairs from
# both the sorted arrays whose sum
# is equal to a given value
def countPairs(arr1, arr2, m, n, x):
    count = 0

    # generating pairs from both
    # the arrays
    for i in range(m):
        for j in range(n):

            # if sum of pair is equal
            # to 'x' increment count
            if arr1[i] + arr2[j] == x:
                count = count + 1

    # required count of pairs
    return count

# Driver Program
arr1 = [1, 3, 5, 7]
arr2 = [2, 3, 5, 8]
m = len(arr1)
n = len(arr2)
x = 10
print("Count = ",
        countPairs(arr1, arr2, m, n, x))

# This code is contributed by Shrikant13.
```

C#

```
 // C# implementation to count pairs from
// both sorted arrays whose sum is equal
// to a given value
using System;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1,
            int []arr2, int m, int n, int x)
```

```
    {
        int count = 0;

        // generating pairs from
        // both the arrays
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)

                // if sum of pair is equal
                // to 'x' increment count
                if ((arr1[i] + arr2[j]) == x)
                    count++;

        // required count of pairs
        return count;
    }

    // Driver Code

    public static void Main ()
    {
        int []arr1 = {1, 3, 5, 7};
        int []arr2 = {2, 3, 5, 8};
        int m = arr1.Length;
        int n = arr2.Length;
        int x = 10;

        Console.WriteLine( "Count = "
        + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value


// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
function countPairs( $arr1, $arr2,
```

```
                    $m, $n, $x)
{
    $count = 0;

    // generating pairs from
    // both the arrays
    for ( $i = 0; $i < $m; $i++)
        for ( $j = 0; $j < $n; $j++)

            // if sum of pair is equal
            // to 'x' increment count
            if (($arr1[$i] + $arr2[$j]) == $x)
                $count++;

    // required count of pairs
    return $count;
}

// Driver Code
$arr1 = array(1, 3, 5, 7);
$arr2 = array(2, 3, 5, 8);
$m = count($arr1);
$n = count($arr2);
$x = 10;
echo "Count = ",
      countPairs($arr1, $arr2,
                  $m,$n, $x);

// This code is contributed by anuj_67.
?>
```

Output :

```
Count = 2
```

Time Complexity : O(mn)
Auxiliary space : O(1)

Method 2 (Binary Search): For each element **arr1[i]**, where **1 <= i <= m**, search the value **(x – arr1[i])** in **arr2[]**. If search is successful, increment the **count**.

C++

```
 // C++ implementation to count
// pairs from both sorted arrays
```

```
// whose sum is equal to a given
// value
#include <bits/stdc++.h>
using namespace std;

// function to search 'value'
// in the given array 'arr[]'
// it uses binary search technique
// as  'arr[]' is sorted
bool isPresent(int arr[], int low,
               int high, int value)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        // value found
        if (arr[mid] == value)
            return true;

        else if (arr[mid] > value)
            high = mid - 1;
        else
            low = mid + 1;
    }

    // value not found
    return false;
}

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;
    for (int i = 0; i < m; i++)
    {
        // for each arr1[i]
        int value = x - arr1[i];

        // check if the 'value'
        // is present in 'arr2[]'
        if (isPresent(arr2, 0, n - 1, value))
            count++;
    }
```

```
    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    int x = 10;
    cout << "Count = "
         << countPairs(arr1, arr2, m, n, x);
    return 0;
}
```

Java

```
 // Java implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
import java.io.*;
class GFG {

// function to search 'value'
// in the given array 'arr[]'
// it uses binary search technique
// as 'arr[]' is sorted
static boolean isPresent(int arr[], int low,
                         int high, int value)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        // value found
        if (arr[mid] == value)
            return true;

        else if (arr[mid] > value)
            high = mid - 1;
        else
            low = mid + 1;
    }
```

```
    // value not found
    return false;
}

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
static int countPairs(int arr1[], int arr2[],
                      int m, int n, int x)
{
    int count = 0;
    for (int i = 0; i < m; i++)
    {

        // for each arr1[i]
        int value = x - arr1[i];

        // check if the 'value'
        // is present in 'arr2[]'
        if (isPresent(arr2, 0, n - 1, value))
            count++;
    }

    // required count of pairs
    return count;
}

    // Driver Code
    public static void main (String[] args)
    {
        int arr1[] = {1, 3, 5, 7};
        int arr2[] = {2, 3, 5, 8};
        int m = arr1.length;
        int n = arr2.length;
        int x = 10;
        System.out.println("Count = "
              + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# implementation to count pairs from both
// sorted arrays whose sum is equal to a given
// value
```

```
using System;

class GFG {

    // function to search 'value' in the given
    // array 'arr[]' it uses binary search
    // technique as 'arr[]' is sorted
    static bool isPresent(int []arr, int low,
                          int high, int value)
    {
        while (low <= high)
        {
            int mid = (low + high) / 2;

            // value found
            if (arr[mid] == value)
                return true;

            else if (arr[mid] > value)
                high = mid - 1;
            else
                low = mid + 1;
        }

        // value not found
        return false;
    }

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1, int []arr2,
                              int m, int n, int x)
    {
        int count = 0;

        for (int i = 0; i < m; i++)
        {

            // for each arr1[i]
            int value = x - arr1[i];

            // check if the 'value'
            // is present in 'arr2[]'
            if (isPresent(arr2, 0, n - 1, value))
                count++;
        }
```

```
        // required count of pairs
        return count;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr1 = {1, 3, 5, 7};
        int []arr2 = {2, 3, 5, 8};
        int m = arr1.Length;
        int n = arr2.Length;
        int x = 10;
        Console.WriteLine("Count = "
            + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value

// function to search 'value'
// in the given array 'arr[]'
// it uses binary search technique
// as 'arr[]' is sorted
function isPresent($arr, $low,
                   $high, $value)
{
    while ($low <= $high)
    {
        $mid = ($low + $high) / 2;

        // value found
        if ($arr[$mid] == $value)
            return true;

        else if ($arr[$mid] > $value)
            $high = $mid - 1;
        else
            $low = $mid + 1;
    }
```

```
    // value not found
    return false;
}

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
function countPairs($arr1, $arr2,
                    $m, $n, $x)
{
    $count = 0;
    for ($i = 0; $i < $m; $i++)
    {

        // for each arr1[i]
        $value = $x - $arr1[$i];

        // check if the 'value'
        // is present in 'arr2[]'
        if (isPresent($arr2, 0,
                      $n - 1, $value))
            $count++;
    }

    // required count of pairs
    return $count;
}

    // Driver Code
    $arr1 = array(1, 3, 5, 7);
    $arr2 = array(2, 3, 5, 8);
    $m = count($arr1);
    $n = count($arr2);
    $x = 10;
    echo "Count = "
        , countPairs($arr1, $arr2, $m, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output :

```
Count = 2
```

Time Complexity : O(mlogn), searching should be applied on the array which is of greater

size so as to reduce the time complexity.
Auxiliary space : O(1)

Method 3 (Hashing): Hash table is implemented using unordered_set in C++. We store all first array elements in hash table. For elements of second array, we subtract every element from **x** and check the result in hash table. If result is present, we increment the **count**.

C++

```
 // C++ implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
#include <bits/stdc++.h>
using namespace std;

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;

    unordered_set<int> us;

    // insert all the elements
    // of 1st array in the hash
    // table(unordered_set 'us')
    for (int i = 0; i < m; i++)
        us.insert(arr1[i]);

    // for each element of 'arr2[]
    for (int j = 0; j < n; j++)

        // find (x - arr2[j]) in 'us'
        if (us.find(x - arr2[j]) != us.end())
            count++;

    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
```

```
    int arr2[] = {2, 3, 5, 8};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    int x = 10;
    cout << "Count = "
         << countPairs(arr1, arr2, m, n, x);
    return 0;
}
```

Output :

```
Count = 2
```

Time Complexity : O(m+n)
Auxiliary space : O(m), hash table should be created of the array having smaller size so as to reduce the space complexity.

Method 4 (Efficient Approach): This approach uses the concept of two pointers, one to traverse 1st array from left to right and another to traverse the 2nd array from right to left.

Algorithm :

```
countPairs(arr1, arr2, m, n, x)

     Initialize l = 0, r = n - 1
     Initialize count = 0

     loop while l = 0
        if (arr1[l] + arr2[r]) == x
           l++, r--
           count++
        else if (arr1[l] + arr2[r]) < x
           l++
        else
           r--

     return count
```

C++

```
 // C++ implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
```

```
#include <bits/stdc++.h>
using namespace std;

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;
    int l = 0, r = n - 1;

    // traverse 'arr1[]' from
    // left to right
    // traverse 'arr2[]' from
    // right to left
    while (l < m && r >= 0)
    {
        // if this sum is equal
        // to 'x', then increment 'l',
        // decrement 'r' and
        // increment 'count'
        if ((arr1[l] + arr2[r]) == x)
        {
            l++; r--;
            count++;
        }

        // if this sum is less
        // than x, then increment l
        else if ((arr1[l] + arr2[r]) < x)
            l++;

        // else decrement 'r'
        else
            r--;
    }

    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
```

```
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    int x = 10;
    cout << "Count = "
          << countPairs(arr1, arr2, m, n, x);
    return 0;
}
```

Java

```
 // Java implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
import java.io.*;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int arr1[],
         int arr2[], int m, int n, int x)
    {
        int count = 0;
        int l = 0, r = n - 1;

        // traverse 'arr1[]' from
        // left to right
        // traverse 'arr2[]' from
        // right to left
        while (l < m && r >= 0)
        {

            // if this sum is equal
            // to 'x', then increment 'l',
            // decrement 'r' and
            // increment 'count'
            if ((arr1[l] + arr2[r]) == x)
            {
                l++; r--;
                count++;
            }

            // if this sum is less
            // than x, then increment l
            else if ((arr1[l] + arr2[r]) < x)
```

```
                l++;

            // else decrement 'r'
            else
                r--;
        }

        // required count of pairs
        return count;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int arr1[] = {1, 3, 5, 7};
        int arr2[] = {2, 3, 5, 8};
        int m = arr1.length;
        int n = arr2.length;
        int x = 10;
        System.out.println( "Count = "
         + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
using System;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1,
        int []arr2, int m, int n, int x)
    {
        int count = 0;
        int l = 0, r = n - 1;

        // traverse 'arr1[]' from
        // left to right
```

```
        // traverse 'arr2[]' from
        // right to left
        while (l < m && r >= 0)
        {

            // if this sum is equal
            // to 'x', then increment 'l',
            // decrement 'r' and
            // increment 'count'
            if ((arr1[l] + arr2[r]) == x)
            {
                l++; r--;
                count++;
            }

            // if this sum is less
            // than x, then increment l
            else if ((arr1[l] + arr2[r]) < x)
                l++;

            // else decrement 'r'
            else
                r--;
        }

        // required count of pairs
        return count;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr1 = {1, 3, 5, 7};
        int []arr2 = {2, 3, 5, 8};
        int m = arr1.Length;
        int n = arr2.Length;
        int x = 10;
        Console.WriteLine( "Count = "
        + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP implementation to count
```

```
// pairs from both sorted arrays
// whose sum is equal to a given
// value


// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
function  countPairs( $arr1,  $arr2,
          $m,  $n,  $x)
{
     $count = 0;
     $l = 0; $r = $n - 1;

    // traverse 'arr1[]' from
    // left to right
    // traverse 'arr2[]' from
    // right to left
    while ($l < $m and $r >= 0)
    {
        // if this sum is equal
        // to 'x', then increment 'l',
        // decrement 'r' and
        // increment 'count'
        if (($arr1[$l] + $arr2[$r]) == $x)
        {
            $l++; $r--;
            $count++;
        }

        // if this sum is less
        // than x, then increment l
        else if (($arr1[$l] + $arr2[$r]) < $x)
            $l++;

        // else decrement 'r'
        else
            $r--;
    }

    // required count of pairs
    return $count;
}

// Driver Code
     $arr1 = array(1, 3, 5, 7);
     $arr2 = array(2, 3, 5, 8);
```

```
    $m = count($arr1);
    $n = count($arr2);
    $x = 10;
    echo "Count = "
   , countPairs($arr1, $arr2, $m, $n, $x);
// This code is contributed by anuj_67

?>
```

Output :

```
Count = 2
```

Time Complexity : O(m + n)
Auxiliary space : O(1)

Improved By : shrikanth13, vt_m

Source

https://www.geeksforgeeks.org/count-pairs-two-sorted-arrays-whose-sum-equal-given-value-x/

Chapter 50

Count pairs from two sorted matrices with given sum

Count pairs from two sorted matrices with given sum - GeeksforGeeks

Given two sorted matrices **mat1** and **mat2** of size **n x n** of distinct elements. Given a value **x**. The problem is to count all pairs from both matrices whose sum is equal to **x**.
Note: The pair has an element from each matrix. Matrices are strictly sorted which means that matrices are sorted in a way such that all elements in a row are sorted in increasing order and for row 'i', where 1 <= i <= n-1, first element of row 'i' is greater than the last element of row 'i-1'.

Examples:

```
Input : mat1[][] = { {1, 5, 6},
                     {8, 10, 11},
                    {15, 16, 18} }

    mat2[][] = { {2, 4, 7},
                 {9, 10, 12},
                 {13, 16, 20} }
       x = 21
Output : 4
The pairs are:
(1, 20), (5, 16), (8, 13) and (11, 10).
```

Method 1 (Naive Approach): For each element **ele** of mat1[][] linearly search **(x − ele)** in mat2[][].

C++

```
// C++ implementation to count pairs from two
```

```
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to search 'val' in mat[][]
// returns true if 'val' is present
// else false
bool valuePresent(int mat[][SIZE], int n, int val)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (mat[i][j] == val)

                // 'val' found
                return true;

    // 'val' not found
    return false;
}

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
               int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)

            // if value (x-mat1[i][j]) is found in mat2[][]
            if (valuePresent(mat2, n, x - mat1[i][j]))
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };
```

```
    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
 // java implementation to count
// pairs from twosorted matrices
// whose sum is equal to a given value
import java.io.*;

class GFG
{
    int SIZE= 10;

    // function to search 'val' in mat[][]
    // returns true if 'val' is present
    // else false
    static boolean valuePresent(int mat[][], int n,
                                            int val)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i][j] == val)

                    // 'val' found
                    return true;

        // 'val' not found
        return false;
    }

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int mat1[][], int mat2[][],
                                        int n, int x)
    {
```

```
        int count = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is
                // found in mat2[][]
                if (valuePresent(mat2, n, x - mat1[i][j]))
                   count++;
            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void main (String[] args)
    {
        int mat1[][] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

        int mat2[][] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        System.out.println ("Count = " +
                            countPairs(mat1, mat2, n, x));

    }
}

// This article is contributed by vt_m
```

Python3

```
 # Python3 implementation to count pairs
# from two sorted matrices whose sum is
# equal to a given value x

# function to search 'val' in mat[][]
# returns true if 'val' is present else
# false
def valuePresent(mat, n, val):

    for i in range(0, n):
```

```
        for j in range(0, n):

            if mat[i][j] == val:

                # 'val' found
                return True

    # 'val' not found
    return False

# function to count pairs from two sorted
# matrices whose sum is equal to a given
# value x
def countPairs(mat1, mat2, n, x):

    count = 0

    for i in range(0, n):
        for j in range(0, n):

            # if value (x-mat1[i][j]) is found
            # in mat2[][]
            if valuePresent(mat2, n, x - mat1[i][j]):
                count += 1

    # required count of pairs
    return count

# Driver program
mat1 = [[ 1, 5, 6 ],
        [ 8, 10, 11 ],
        [ 15, 16, 18 ] ]

mat2 = [ [ 2, 4, 7 ],
         [ 9, 10, 12 ],
         [ 13, 16, 20 ] ]

n = 3
x = 21

print( "Count = "),
print(countPairs(mat1, mat2, n, x))

# This code is contributed by upendra bartwal
```

C#

```
 //C# implementation to count
```

```
// pairs from twosorted matrices
// whose sum is equal to a given value
using System;

class GFG
{
    // int SIZE= 10;

    // function to search 'val' in mat[][]
    // returns true if 'val' is present
    // else false
    static bool valuePresent(int[,] mat, int n,
                                        int val)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i, j] == val)

                    // 'val' found
                    return true;

        // 'val' not found
        return false;
    }

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int [,]mat1, int [,]mat2,
                                        int n, int x)
    {
        int count = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is
                // found in mat2[][]
                if (valuePresent(mat2, n, x - mat1[i,j]))
                count++;
            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void Main ()
    {
```

```
        int [,]mat1 = { { 1, 5, 6 },
                        { 8, 10, 11 },
                        { 15, 16, 18 } };

        int [,]mat2 = { { 2, 4, 7 },
                        { 9, 10, 12 },
                        { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        Console.WriteLine("Count = " +
                        countPairs(mat1, mat2, n, x));

    }
}

// This article is contributed by vt_m
```

Output:

```
Count = 4
```

Time Complexity: O(n^4).
Auxiliary Space: O(1).

Method 2 (Binary Search): As matrix is strictly sorted, use the concept of binary search technique. For each element **ele** of mat1[][] apply the binary search technique on the elements of the first column of mat2[][] to find the row index number of the largest element smaller than equal to **(x − ele)**. Let it be **row_no**. If no such row exists then no pair can be formed with element **ele**. Else apply the concept of binary search technique to find the value **(x − ele)** in the row represented by **row_no** in mat2[][]. If value found then increment **count**.

C++

```
 // C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a given
// value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function returns the row index no of largest
// element smaller than equal to 'x' in first
```

```
// column of mat[][]. If no such element exists
// then it returns -1.
int binarySearchOnRow(int mat[SIZE][SIZE],
                      int l, int h, int x)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or equal to mat[mid][0],
        // then search in mat[mid+1...h][0]
        if (mat[mid][0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function to search 'val' in mat[row][]
bool binarySearchOnCol(int mat[][SIZE], int l, int h,
                       int val, int row)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row][mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row][mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}

// function to search 'val' in mat[][]
// returns true if 'val' is present
```

```
// else false
bool searchValue(int mat[][SIZE],
                 int n, int val)
{
    // to get the row index number of the largest element
    // smaller than equal to 'val' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no][]
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
               int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            // if value (x-mat1[i][j]) is found in mat2[][]
            if (searchValue(mat2, n, x - mat1[i][j]))
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;
```

```
    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
 // java implementation to count
// pairs from two sorted matrices
// whose sum is equal to a given
// value x
import java.io.*;

class GFG {
    int SIZE= 10;

    // function returns the row index no of largest
    // element smaller than equal to 'x' in first
    // column of mat[][]. If no such element exists
    // then it returns -1.
    static int binarySearchOnRow(int mat[][], int l,
                                     int h, int x)
    {
        while (l <= h)
        {
            int mid = (l + h) / 2;

            // if 'x' is greater than or
            // equal to mat[mid][0], then
            // search in mat[mid+1...h][0]
            if (mat[mid][0] <= x)
                l = mid + 1;

            // else search in mat[l...mid-1][0]
            else
                h = mid - 1;
        }

        // required row index number
        return h;
    }

    // function to search 'val' in mat[row][]
    static boolean binarySearchOnCol(int mat[][], int l, int h,
                                            int val, int row)
    {
        while (l <= h)
```

```
    {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row][mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row][mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}

// function to search 'val' in mat[][]
// returns true if 'val' is present
// else false
static boolean searchValue(int mat[][],
                           int n, int val)
{
    // to get the row index number
    // of the largest element smaller
    // than equal to 'val' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no][]
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from
// two sorted matrices whose sum
// is equal to a given value x
static int countPairs(int mat1[][], int mat2[][],
                                    int n, int x)
{
    int count = 0;
```

```
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is found in mat2[][]
                if (searchValue(mat2, n, x - mat1[i][j]))
                    count++;
            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void main (String[] args)
    {
        int mat1[][] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

        int mat2[][] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        System.out.println ( "Count = "  +
                           countPairs(mat1, mat2, n, x));

    }
}

// This code is contributed by vt_m
```

C#

```
 // C# implementation to count
// pairs from two sorted matrices
// whose sum is equal to a given
// value x
using System;

class GFG
{
    //int SIZE= 10;

    // function returns the row index no of largest
    // element smaller than equal to 'x' in first
    // column of mat[][]. If no such element exists
```

```
// then it returns -1.
static int binarySearchOnRow(int [,]mat, int l,
                                int h, int x)
{
    while (l <= h)
    {
        int mid = (l + h) / 2;

        // if 'x' is greater than or
        // equal to mat[mid][0], then
        // search in mat[mid+1...h][0]
        if (mat[mid,0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function to search 'val' in mat[row][]
static bool binarySearchOnCol(int [,]mat, int l, int h,
                                        int val, int row)
{
    while (l <= h)
    {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row,mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row,mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}
```

```
// function to search 'val' in mat[][]
// returns true if 'val' is present
// else false
static bool searchValue(int [,]mat,
                        int n, int val)
{
    // to get the row index number
    // of the largest element smaller
    // than equal to 'val' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no][]
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from
// two sorted matrices whose sum
// is equal to a given value x
static int countPairs(int [,]mat1, int [,]mat2,
                                    int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            // if value (x-mat1[i][j]) is found in mat2[][]
            if (searchValue(mat2, n, x - mat1[i,j]))
                count++;
        }
    // required count of pairs
    return count;
}

// Driver program
public static void Main ()
{
    int [,]mat1 = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int [,]mat2 = { { 2, 4, 7 },
                    { 9, 10, 12 },
```

```
                        { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        Console.WriteLine ( "Count = " +
                        countPairs(mat1, mat2, n, x));

    }
}

// This code is contributed by vt_m
```

Output:

```
Count = 4
```

Time Complexity: ($n^2 \log_2 n$).
Auxiliary Space: O(1).

Method 3 (Hashing): Create a hash table and insert all the elements of mat2[][] in it. Now for each element **ele** of mat1[][] find **(x − ele)** in the hash table.

```
 // C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
               int n, int x)
{
    int count = 0;

    // unordered_set 'us' implemented as hash table
    unordered_set<int> us;

    // insert all the elements of mat2[][] in 'us'
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            us.insert(mat2[i][j]);
```

```
    // for each element of mat1[][]
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)

            // if (x-mat1[i][j]) is in 'us'
            if (us.find(x - mat1[i][j]) != us.end())
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Output:

```
Count = 4
```

Time complexity: $O(n^2)$.
Auxiliary Space: $O(n^2)$.

Method 4 (Efficient Approach): From the top leftmost element traverse mat1[][] in forward direction (i.e., from the topmost row up to last, each row is being traversed from left to right) and from the bottom rightmost element traverse mat2[][] in backward direction (i.e, from the bottom row up to first, each row is being traversed from right to left). For each element **e1** of mat1[][] and **e2** of mat2[][] encountered, calculate **val** = (e1 + e2). If val == x, increment **count**. Else if val is less than x, move to next element of mat1[][] in forward direction. Else move to next element of mat2[][] in backward direction. Continue

this process until either of the two matrices gets completely traversed.

C++

```
 // C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
                                    int n, int x)
{
    // 'r1' and 'c1' for pointing current element
    // of mat1[][]
    // 'r2' and 'c2' for pointing current element
    // of mat2[][]
    int r1 = 0, c1 = 0;
    int r2 = n - 1, c2 = n - 1;

    // while there are more elements
    // in both the matrices
    int count = 0;
    while ((r1 < n) && (r2 >= -1)) {
        int val = mat1[r1][c1] + mat2[r2][c2];

        // if true
        if (val == x) {

            // increment 'count'
            count++;

            // move mat1[][] column 'c1' to right
            // move mat2[][] column 'c2' to left
            c1++;
            c2--;
        }

        // if true, move mat1[][] column 'c1' to right
        else if (val < x)
            c1++;

        // else move mat2[][] column 'c2' to left
```

```
        else
            c2--;

        // if 'c1' crosses right boundary
        if (c1 == n) {

            // reset 'c1'
            c1 = 0;

            // increment row 'r1'
            r1++;
        }

        // if 'c2' crosses left boundary
        if (c2 == -1) {

            // reset 'c2'
            c2 = n - 1;

            // decrement row 'r2'
            r2--;
        }
    }

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
 // java implementation to count
// pairs from two  sorted
// matrices whose sum is
// equal to agiven value x
import java.io.*;

class GFG
{
    int SIZE = 10;

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int mat1[][], int mat2[][],
                                        int n, int x)
    {
        // 'r1' and 'c1' for pointing current
        // element of mat1[][]
        // 'r2' and 'c2' for pointing current
        // element of mat2[][]
        int r1 = 0, c1 = 0;
        int r2 = n - 1, c2 = n - 1;

        // while there are more elements
        // in both the matrices
        int count = 0;
        while ((r1 < n) && (r2 >= -1))
        {
            int val = mat1[r1][c1] + mat2[r2][c2];

            // if true
            if (val == x) {

                // increment 'count'
                count++;

                // move mat1[][] column 'c1' to right
                // move mat2[][] column 'c2' to left
                c1++;
                c2--;
            }

            // if true, move mat1[][]
            // column 'c1' to right
            else if (val < x)
                c1++;
```

```
        // else move mat2[][] column
        // 'c2' to left
        else
            c2--;

        // if 'c1' crosses right boundary
        if (c1 == n) {

            // reset 'c1'
            c1 = 0;

            // increment row 'r1'
            r1++;
        }

        // if 'c2' crosses left boundary
        if (c2 == -1) {

            // reset 'c2'
            c2 = n - 1;

            // decrement row 'r2'
            r2--;
        }
    }

    // required count of pairs
    return count;
}

// Driver code
public static void main (String[] args)
{
    int mat1[][] = { { 1, 5, 6 },
                     { 8, 10, 11 },
                     { 15, 16, 18 } };

    int mat2[][] = { { 2, 4, 7 },
                     { 9, 10, 12 },
                     { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    System.out.println ( "Count = " +
                         countPairs(mat1, mat2, n, x));
```

```
    }
}

// This article is contributed by vt_m
```

C#

```
 // C# implementation to count pairs
// from two sorted matrices whose
// sum is equal to a given value x
using System;

class GFG {

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int [,]mat1,
            int [,]mat2, int n, int x)
    {

        // 'r1' and 'c1' for pointing
        // current element of mat1[][]
        // 'r2' and 'c2' for pointing
        // current element of mat2[][]
        int r1 = 0, c1 = 0;
        int r2 = n - 1, c2 = n - 1;

        // while there are more elements
        // in both the matrices
        int count = 0;
        while ((r1 < n) && (r2 >= -1))
        {
            int val = mat1[r1,c1]
                          + mat2[r2,c2];

            // if true
            if (val == x) {

                // increment 'count'
                count++;

                // move mat1[][] column
                // 'c1' to right
                // move mat2[][] column
                // 'c2' to left
                c1++;
                c2--;
```

```
        }

        // if true, move mat1[][]
        // column 'c1' to right
        else if (val < x)
            c1++;

        // else move mat2[][] column
        // 'c2' to left
        else
            c2--;

        // if 'c1' crosses right
        // boundary
        if (c1 == n) {

            // reset 'c1'
            c1 = 0;

            // increment row 'r1'
            r1++;
        }

        // if 'c2' crosses left
        // boundary
        if (c2 == -1) {

            // reset 'c2'
            c2 = n - 1;

            // decrement row 'r2'
            r2--;
        }
    }

    // required count of pairs
    return count;
}

// Driver code
public static void Main ()
{
    int [,]mat1 = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int [,]mat2 = { { 2, 4, 7 },
                    { 9, 10, 12 },
```

```
                        { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        Console.Write ( "Count = " +
            countPairs(mat1, mat2, n, x));

    }
}

// This code is contributed by
// nitin mittal
```

Output:

```
Count = 4
```

Time Complexity: O(n^2).
Auxiliary Space: O(1).

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-pairs-two-sorted-matrices-given-sum/

Chapter 51

Count pairs whose products exist in array

Count pairs whose products exist in array - GeeksforGeeks

Given an array, count those pair whose product value is present in array.

Examples:

```
Input : arr[] = {6, 2, 4, 12, 5, 3}
Output : 3
       All pairs whose product exist in array
       (6 , 2) (2, 3) (4, 3)

Input :  arr[] = {3, 5, 2, 4, 15, 8}
Output : 2
```

A **Simple solution** is to generate all pairs of given array and check if product exists in the array. If exists, then increment count. Finally return count.

Below is implementation of above idea

C++

```
 // C++ program to count pairs whose product exist in array
#include<iostream>
using namespace std;

// Returns count of pairs whose product exists in arr[]
int countPairs( int arr[] ,int n)
{
    int result = 0;
```

```
    for (int i = 0; i < n ; i++)
    {
        for (int j = i+1 ; j < n ; j++)
        {
            int product = arr[i] * arr[j] ;

            // find product in an array
            for (int k = 0; k < n; k++)
            {
                // if product found increment counter
                if (arr[k] == product)
                {
                    result++;
                    break;
                }
            }
        }
    }

    // return Count of all pair whose product exist in array
    return result;
}

//Driver program
int main()
{
    int arr[] = {6 ,2 ,4 ,12 ,5 ,3} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n);
    return 0;
}
```

Java

```
 // Java program to count pairs
// whose product exist in array
import java.io.*;

class GFG
{

// Returns count of pairs
// whose product exists in arr[]
static int countPairs(int arr[],
                      int n)
{
    int result = 0;
    for (int i = 0; i < n ; i++)
```

```
    {
        for (int j = i + 1 ; j < n ; j++)
        {
            int product = arr[i] * arr[j] ;

            // find product
            // in an array
            for (int k = 0; k < n; k++)
            {
                // if product found
                // increment counter
                if (arr[k] == product)
                {
                    result++;
                    break;
                }
            }
        }
    }

    // return Count of all pair
    // whose product exist in array
    return result;
}

// Driver Code
public static void main (String[] args)
{
int arr[] = {6, 2, 4, 12, 5, 3} ;
int n = arr.length;
System.out.println(countPairs(arr, n));
}
}

// This code is contributed by anuj_67.
```

Output:

```
3
```

Time complexity: $O(n^3)$

An **Efficient solution** is to use 'hash' that stores all array element. Generate all possible pair of given array 'arr' and check product of each pair is in 'hash'. If exists, then increment count. Finarlly return count.

Below is implementation of above idea

C++

```
 // A hashing based C++ program to count pairs whose product
// exists in arr[]
#include<bits/stdc++.h>
using namespace std;

// Returns count of pairs whose product exists in arr[]
int countPairs(int arr[] , int n)
{
    int result = 0;

    // Create an empty hash-set that store all array element
    set< int > Hash;

    // Insert all array element into set
    for (int i = 0 ; i < n; i++)
        Hash.insert(arr[i]);

    // Generate all pairs and check is exist in 'Hash' or not
    for (int i = 0 ; i < n; i++)
    {
        for (int j = i + 1; j<n ; j++)
        {
            int product = arr[i]*arr[j];

            // if product exists in set then we increment
            // count by 1
            if (Hash.find(product) != Hash.end())
                result++;
        }
    }

    // return count of pairs whose product exist in array
    return result;
}

// Driver program
int main()
{
    int arr[] = {6 ,2 ,4 ,12 ,5 ,3};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n) ;
    return 0;
}
```

Output:

```
3
```

Time complexity : O(n^2) 'Under the assumption insert, find operation take O(1) Time '

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-pairs-whose-products-exist-in-array/

Chapter 52

Count pairs with given sum

Count pairs with given sum - GeeksforGeeks

Given an array of integers, and a number 'sum', find the number of pairs of integers in the array whose sum is equal to 'sum'.

```
Examples:
Input  :  arr[] = {1, 5, 7, -1},
          sum = 6
Output :  2
Pairs with sum 6 are (1, 5) and (7, -1)

Input  :  arr[] = {1, 5, 7, -1, 5},
          sum = 6
Output :  3
Pairs with sum 6 are (1, 5), (7, -1) &
                     (1, 5)

Input  :  arr[] = {1, 1, 1, 1},
          sum = 2
Output :  6
There are 3! pairs with sum 2.

Input  :  arr[] = {10, 12, 10, 15, -1, 7, 6,
                   5, 4, 2, 1, 1, 1},
          sum = 11
Output :  9
```

Expected time complexity O(n)

A **simple solution** is be traverse each element and check if there's another number in the array which can be added to it to give sum.

C++

```
 // C++ implementation of simple method to find count of
// pairs with given sum.
#include <bits/stdc++.h>
using namespace std;

// Returns number of pairs in arr[0..n-1] with sum equal
// to 'sum'
int getPairsCount(int arr[], int n, int sum)
{
    int count = 0; // Initialize result

    // Consider all possible pairs and check their sums
    for (int i=0; i<n; i++)
        for (int j=i+1; j<n; j++)
            if (arr[i]+arr[j] == sum)
                count++;

    return count;
}

// Driver function to test the above function
int main()
{
    int arr[] = {1, 5, 7, -1, 5} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = 6;
    cout << "Count of pairs is "
         << getPairsCount(arr, n, sum);
    return 0;
}
```

Java

```
 // Java implementation of simple method to find count of
// pairs with given sum.
public class find
{
    public static void main(String args[])
    {
        int[] arr = { 1, 5, 7, -1, 5 };
        int sum = 6;
        getPairsCount(arr, sum);
    }

    // Prints number of pairs in arr[0..n-1] with sum equal
```

```
    // to 'sum'
    public static void getPairsCount(int[] arr, int sum)
    {

        int count = 0;// Initialize result

        // Consider all possible pairs and check their sums
        for (int i = 0; i < arr.length; i++)
            for (int j = i + 1; j < arr.length; j++)
                if ((arr[i] + arr[j]) == sum)
                    count++;

        System.out.printf("Count of pairs is %d",count);
    }
}
// This program is contributed by Jyotsna
```

Python3

```
 # Python3 implementation of simple method
# to find count of pairs with given sum.

# Returns number of pairs in arr[0..n-1]
# with sum equal to 'sum'
def getPairsCount(arr, n, sum):

    count = 0 # Initialize result

    # Consider all possible pairs
    # and check their sums
    for i in range(0, n):
        for j in range(i + 1, n):
            if arr[i] + arr[j] == sum:
                count += 1

    return count

# Driver function
arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6
print("Count of pairs is",
      getPairsCount(arr, n, sum))

# This code is contributed by Smitha Dinesh Semwal
```

PHP

```
 <?php
// PHP implementation of simple
// method to find count of
// pairs with given sum.

// Returns number of pairs in
// arr[0..n-1] with sum equal
// to 'sum'
function getPairsCount($arr, $n, $sum)
{
    // Initialize result
    $count = 0;

    // Consider all possible pairs
    // and check their sums
    for ($i = 0; $i < $n; $i++)
        for ($j = $i + 1; $j < $n; $j++)
            if ($arr[$i] + $arr[$j] == $sum)
                $count++;

    return $count;
}

    // Driver Code
    $arr = array(1, 5, 7, -1, 5) ;
    $n = sizeof($arr);
    $sum = 6;
    echo "Count of pairs is "
         , getPairsCount($arr, $n, $sum);

// This code is contributed by nitin mittal.
?>
```

Output :

```
Count of pairs is 3
```

Time Complexity : O(n^2)
Auxiliary Space : O(1)

A better solution is possible in O(n) time.

Below is the Algorithm.

1. Create a map to store frequency of each number in the array. (Single traversal is required)
2. In the next traversal, for every element check if it can be combined with any other element (other than itself!) to give the desired sum. Increment the counter accordingly.

3. After completion of second traversal, we'd have twice the required value stored in counter because every pair is counted two times. Hence divide count by 2 and return.

Below is the implementation of above idea :

C++

```
 // C++ implementation of simple method to find count of
// pairs with given sum.
#include <bits/stdc++.h>
using namespace std;

// Returns number of pairs in arr[0..n-1] with sum equal
// to 'sum'
int getPairsCount(int arr[], int n, int sum)
{
    unordered_map<int, int> m;

    // Store counts of all elements in map m
    for (int i=0; i<n; i++)
        m[arr[i]]++;

    int twice_count = 0;

    // iterate through each element and increment the
    // count (Notice that every pair is counted twice)
    for (int i=0; i<n; i++)
    {
        twice_count += m[sum-arr[i]];

        // if (arr[i], arr[i]) pair satisfies the condition,
        // then we need to ensure that the count is
        // decreased by one such that the (arr[i], arr[i])
        // pair is not considered
        if (sum-arr[i] == arr[i])
            twice_count--;
    }

    // return the half of twice_count
    return twice_count/2;
}

// Driver function to test the above function
int main()
{
    int arr[] = {1, 5, 7, -1, 5} ;
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    int sum = 6;
    cout << "Count of pairs is "
         << getPairsCount(arr, n, sum);
    return 0;
}
```

Java

```
 /* Java implementation of simple method to find count of
pairs with given sum*/

import java.util.HashMap;

class Test
{
    static int arr[] = new int[]{1, 5, 7, -1, 5} ;

    // Returns number of pairs in arr[0..n-1] with sum equal
    // to 'sum'
    static int getPairsCount(int n, int sum)
    {
        HashMap<Integer, Integer> hm = new HashMap<>();

        // Store counts of all elements in map hm
        for (int i=0; i<n; i++){

            // initializing value to 0, if key not found
            if(!hm.containsKey(arr[i]))
                hm.put(arr[i],0);

            hm.put(arr[i], hm.get(arr[i])+1);
        }
        int twice_count = 0;

        // iterate through each element and increment the
        // count (Notice that every pair is counted twice)
        for (int i=0; i<n; i++)
        {
            if(hm.get(sum-arr[i]) != null)
                twice_count += hm.get(sum-arr[i]);

            // if (arr[i], arr[i]) pair satisfies the condition,
            // then we need to ensure that the count is
            // decreased by one such that the (arr[i], arr[i])
            // pair is not considered
            if (sum-arr[i] == arr[i])
                twice_count--;
        }
```

```
        // return the half of twice_count
        return twice_count/2;
    }

    // Driver method to test the above function
    public static void main(String[] args) {

        int sum = 6;
        System.out.println("Count of pairs is " +
                           getPairsCount(arr.length,sum));

    }
}
// This code is contributed by Gaurav Miglani
```

Python3

```
 # Python 3 implementation of simple method
# to find count of pairs with given sum.
import sys

# Returns number of pairs in arr[0..n-1]
# with sum equal to 'sum'
def getPairsCount(arr, n, sum):

    m = [0] * 1000

    # Store counts of all elements in map m
    for i in range(0, n):
        m[arr[i]]
        m[arr[i]] += 1

    twice_count = 0

    # Iterate through each element and increment
    # the count (Notice that every pair is
    # counted twice)
    for i in range(0, n):

        twice_count += m[sum - arr[i]]

        # if (arr[i], arr[i]) pair satisfies the
        # condition, then we need to ensure that
        # the count is  decreased by one such
        # that the (arr[i], arr[i]) pair is not
        # considered
        if (sum - arr[i] == arr[i]):
```

```
            twice_count -= 1

    # return the half of twice_count
    return int(twice_count / 2)

# Driver function
arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6

print("Count of pairs is", getPairsCount(arr,
                                         n, sum))

# This code is contributed by
# Smitha Dinesh Semwal
```

Output :

```
Count of pairs is 3
```

This article is contributed by Ashutosh Kumar. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-pairs-with-given-sum/

Chapter 53

Count quadruples from four sorted arrays whose sum is equal to a given value x

Count quadruples from four sorted arrays whose sum is equal to a given value x - GeeksforGeeks

Given four sorted arrays each of size **n** of distinct elements. Given a value **x**. The problem is to count all **quadruples**(group of four numbers) from all the four arrays whose sum is equal to **x**.

Note: The quadruple has an element from each of the four arrays.

Examples:

```
Input : arr1 = {1, 4, 5, 6},
        arr2 = {2, 3, 7, 8},
        arr3 = {1, 4, 6, 10},
        arr4 = {2, 4, 7, 8}
        n = 4, x = 30

Output : 4
The quadruples are:
(4, 8, 10, 8), (5, 7, 10, 8),
(5, 8, 10, 7), (6, 7, 10, 7)

Input : For the same above given fours arrays
        x = 25
Output : 14
```

Method 1 (Naive Approach): Using four nested loops generate all quadruples and check whether elements in the quadruple sum up to **x** or not.

C++

```
// C++ implementation to count quadruples from four sorted arrays
// whose sum is equal to a given value x
#include <bits/stdc++.h>

using namespace std;

// function to count all quadruples from
// four sorted arrays whose sum is equal
// to a given value x
int countQuadruples(int arr1[], int arr2[],
                    int arr3[], int arr4[], int n, int x)
{
    int count = 0;

    // generate all possible quadruples from
    // the four sorted arrays
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                for (int l = 0; l < n; l++)
                    // check whether elements of
                    // quadruple sum up to x or not
                    if ((arr1[i] + arr2[j] + arr3[k] + arr4[l]) == x)
                        count++;

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3,
                            arr4, n, x);
    return 0;
}
```

Python3

```
 # A Python implementation to count
# quadruples from four sorted arrays
# whose sum is equal to a given value x

# function to count all quadruples
# from four sorted arrays whose sum
# is equal to a given value x
def countQuuadruples(arr1, arr2,
                     arr3, arr4, n, x):
    count = 0

    # generate all possible
    # quadruples from the four
    # sorted arrays
    for i in range(n):
        for j in range(n):
            for k in range(n):
                for l in range(n):

                    # check whether elements of
                    # quadruple sum up to x or not
                    if (arr1[i] + arr2[j] +
                        arr3[k] + arr4[l] == x):
                        count += 1

    # required count of quadruples
    return count

# Driver Code
arr1 = [1, 4, 5, 6]
arr2 = [2, 3, 7, 8]
arr3 = [1, 4, 6, 10]
arr4 = [2, 4, 7, 8 ]
n = len(arr1)
x = 30
print("Count = ", countQuuadruples(arr1, arr2,
                                   arr3, arr4, n, x))

# This code is contributed
# by Shrikant13
```

Output:

```
Count = 4
```

Time Complexity: O(n^4)
Auxiliary Space: O(1)

Method 2 (Binary Search): Generate all triplets from the 1st three arrays. For each triplet so generated, find the sum of elements in the triplet. Let it be **T**. Now, search the value **(x − T)** in the 4th array. If value found in the 4th array, then increment **count**. This process is repeated for all the triplets generated from the 1st three arrays.

```
 // C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

// find the 'value' in the given array 'arr[]'
// binary search technique is applied
bool isPresent(int arr[], int low, int high, int value)
{
    while (low <= high) {
        int mid = (low + high) / 2;

        // 'value' found
        if (arr[mid] == value)
            return true;
        else if (arr[mid] > value)
            high = mid - 1;
        else
            low = mid + 1;
    }

    // 'value' not found
    return false;
}

// function to count all quadruples from four
// sorted arrays whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                    int arr4[], int n, int x)
{
    int count = 0;

    // generate all triplets from the 1st three arrays
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++) {

                // calculate the sum of elements in
                // the triplet so generated
```

```
                int T = arr1[i] + arr2[j] + arr3[k];

                // check if 'x-T' is present in 4th
                // array or not
                if (isPresent(arr4, 0, n, x - T))

                    // increment count
                    count++;
            }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3, arr4, n, x);
    return 0;
}
```

Output:

```
Count = 4
```

Time Complexity: O(n^3logn)
Auxiliary Space: O(1)

Method 3 (Use of two pointers): Generate all pairs from the 1st two arrays. For each pair so generated, find the sum of elements in the pair. Let it be **p_sum**. For each **p_sum**, count pairs from the 3rd and 4th sorted array with sum equal to **(x − p_sum)**. Accumulate these count in the **total_count** of quadruples.

```
 // C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
#include <bits/stdc++.h>
```

```
using namespace std;

// count pairs from the two sorted array whose sum
// is equal to the given 'value'
int countPairs(int arr1[], int arr2[], int n, int value)
{
    int count = 0;
    int l = 0, r = n - 1;

    // traverse 'arr1[]' from left to right
    // traverse 'arr2[]' from right to left
    while (l < n & amp; &r >= 0) {
        int sum = arr1[l] + arr2[r];

        // if the 'sum' is equal to 'value', then
        // increment 'l', decrement 'r' and
        // increment 'count'
        if (sum == value) {
            l++, r--;
            count++;
        }

        // if the 'sum' is greater than 'value', then
        // decrement r
        else if (sum > value)
            r--;

        // else increment l
        else
            l++;
    }

    // required count of pairs
    return count;
}

// function to count all quadruples from four sorted arrays
// whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                    int arr4[], int n, int x)
{
    int count = 0;

    // generate all pairs from arr1[] and arr2[]
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            // calculate the sum of elements in
            // the pair so generated
```

```
            int p_sum = arr1[i] + arr2[j];

            // count pairs in the 3rd and 4th array
            // having value 'x-p_sum' and then
            // accumulate it to 'count'
            count += countPairs(arr3, arr4, n, x - p_sum);
        }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3,
                            arr4, n, x);
    return 0;
}
```

Output:

```
Count = 4
```

Time Complexity: O(n^3)
Auxiliary Space: O(1)

Method 4 Efficient Approach(Hashing): Create a hash table where **(key, value)** tuples are represented as **(sum, frequency)** tuples. Here the sum are obtained from the pairs of 1st and 2nd array and their frequency count is maintained in the hash table. Hash table is implemented using unordered_map in C++. Now, generate all pairs from the 3rd and 4th array. For each pair so generated, find the sum of elements in the pair. Let it be **p_sum**. For each **p_sum**, check whether **(x − p_sum)** exists in the hash table or not. If it exists, then add the frequency of **(x − p_sum)** to the **count** of quadruples.

```
 // C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
```

```
#include <bits/stdc++.h>

using namespace std;

// function to count all quadruples from four sorted
// arrays whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                    int arr4[], int n, int x)
{
    int count = 0;

    // unordered_map 'um' implemented as hash table
    // for <sum, frequency> tuples
    unordered_map<int, int> um;

    // count frequency of each sum obtained from the
    // pairs of arr1[] and arr2[] and store them in 'um'
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            um[arr1[i] + arr2[j]]++;

    // generate pair from arr3[] and arr4[]
    for (int k = 0; k < n; k++)
        for (int l = 0; l < n; l++) {

            // calculate the sum of elements in
            // the pair so generated
            int p_sum = arr3[k] + arr4[l];

            // if 'x-p_sum' is present in 'um' then
            // add frequency of 'x-p_sum' to 'count'
            if (um.find(x - p_sum) != um.end())
                count += um[x - p_sum];
        }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };
```

```
    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3, arr4, n, x);
    return 0;
}
```

Output:

```
Count = 4
```

Time Complexity: $O(n^2)$
Auxiliary Space: $O(n^2)$

Improved By : shrikanth13

Source

https://www.geeksforgeeks.org/count-quadruples-four-sorted-arrays-whose-sum-equal-given-value-x/

Chapter 54

Count subarrays having total distinct elements same as original array

Count subarrays having total distinct elements same as original array - GeeksforGeeks

Given an array of n integers. Count total number of sub-array having total distinct elements same as that of total distinct elements of original array.

```
Input  : arr[] = {2, 1, 3, 2, 3}
Output : 5
Total distinct elements in array is 3
Total sub-arrays that satisfy the condition
are:  Subarray from index 0 to 2
      Subarray from index 0 to 3
      Subarray from index 0 to 4
      Subarray from index 1 to 3
      Subarray from index 1 to 4

Input  : arr[] = {2, 4, 5, 2, 1}
Output : 2

Input  : arr[] = {2, 4, 4, 2, 4}
Output : 9
```

A **Naive approach** is to run a loop one inside another and consider all sub-arrays and for every sub-array count all distinct elements by using hashing and compare it with total distinct elements of original array. This approach takes $O(n^2)$ time.

An **efficient approach** is to use sliding window to count all distinct elements in one iteration.

1. Find the number of distinct elements in the entire array. Let this number be **k <= N**. Initialize Left = 0, Right = 0 and window = 0.
2. Increment **right** until the number of distinct elements in range [**Left=0, Right**] equal to **k**(or window size would not equal to k), let this right be $\mathbf{R_1}$. Now since the sub-array [**Left = 0,** $\mathbf{R_1}$] has **k** distinct elements, so all the sub-arrays starting at **Left = 0** and ending after $\mathbf{R_1}$ will also have **k** distinct elements. Thus add **N-**$\mathbf{R_1}$**+1** to the answer because [**Left..** $\mathbf{R_1}$], [**Left..** $\mathbf{R_1}$**+1**], [**Left..** $\mathbf{R_1}$**+2**] **...** [**Left.. N-1**] contains all the distinct numbers.
3. Now keeping $\mathbf{R_1}$ same, increment **left**. Decrease the frequency of the previous element i.e., **arr[0],** and if its frequency becomes 0, decrease the window size. Now, the sub-array is [**Left = 1, Right =** $\mathbf{R_1}$].
4. Repeat the same process from step 2 for other values of Left and Right till **Left < N**.

C++

```
 // C++ program Count total number of sub-arrays
// having total distinct elements same as that
// original array.
#include<bits / stdc++.h>
using namespace std;

// Function to calculate distinct sub-array
int countDistictSubarray(int arr[], int n)
{
    // Count distinct elements in whole array
    unordered_map<int, int>  vis;
    for (int i = 0; i < n; ++i)
        vis[arr[i]] = 1;
    int k = vis.size();

    // Reset the container by removing all elements
    vis.clear();

    // Use sliding window concept to find
    // count of subarrays having k distinct
    // elements.
    int ans = 0, right = 0, window = 0;
    for (int left = 0; left < n; ++left)
    {
        while (right < n && window < k)
        {
            ++vis[ arr[right] ];

            if (vis[ arr[right] ] == 1)
                ++window;

            ++right;
        }
```

```
        // If window size equals to array distinct
        // element size, then update answer
        if (window == k)
            ans += (n - right + 1);

        // Decrease the frequency of previous element
        // for next sliding window
        --vis[ arr[left] ];

        // If frequency is zero then decrease the
        // window size
        if (vis[ arr[left] ] == 0)
                --window;
    }
    return ans;
}

// Driver code
int main()
{
    int arr[] = {2, 1, 3, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << countDistictSubarray(arr, n) <<"n";
    return 0;
}
```

Java

```
 // Java program Count total number of sub-arrays
// having total distinct elements same as that
// original array.

import java.util.HashMap;

class Test
{
    // Method to calculate distinct sub-array
    static int countDistictSubarray(int arr[], int n)
    {
        // Count distinct elements in whole array
        HashMap<Integer, Integer>  vis = new HashMap<Integer,Integer>(){
            @Override
            public Integer get(Object key) {
                if(!containsKey(key))
                    return 0;
                return super.get(key);
```

```
        }
    };

    for (int i = 0; i < n; ++i)
        vis.put(arr[i], 1);
    int k = vis.size();

    // Reset the container by removing all elements
    vis.clear();

    // Use sliding window concept to find
    // count of subarrays having k distinct
    // elements.
    int ans = 0, right = 0, window = 0;
    for (int left = 0; left < n; ++left)
    {
        while (right < n && window < k)
        {
            vis.put(arr[right], vis.get(arr[right]) + 1);

            if (vis.get(arr[right])== 1)
                ++window;

            ++right;
        }

        // If window size equals to array distinct
        // element size, then update answer
        if (window == k)
            ans += (n - right + 1);

        // Decrease the frequency of previous element
        // for next sliding window
        vis.put(arr[left], vis.get(arr[left]) - 1);

        // If frequency is zero then decrease the
        // window size
        if (vis.get(arr[left]) == 0)
                --window;
    }
    return ans;
}

// Driver method
public static void main(String args[])
{
    int arr[] = {2, 1, 3, 2, 3};
```

```
        System.out.println(countDistictSubarray(arr, arr.length));
    }
}
```

```
Output:
5
```

Time complexity: O(n)
Auxiliary space: O(n)

Source

https://www.geeksforgeeks.org/count-subarrays-total-distinct-elements-original-array/

Chapter 55

Count subarrays with equal number of 1's and 0's

Count subarrays with equal number of 1's and 0's - GeeksforGeeks

Given an array **arr[]** of size **n** containing 0 and 1 only. The problem is to count the subarrays having equal number of 0's and 1's.

Examples:

```
Input : arr[] = {1, 0, 0, 1, 0, 1, 1}
Output : 8
The index range for the 8 sub-arrays are:
(0, 1), (2, 3), (0, 3), (3, 4), (4, 5)
(2, 5), (0, 5), (1, 6)
```

The problem is closely related to Largest subarray with equal number of 0's and 1's.

Approach: Following are the steps:

1. Consider all 0's in arr[] as -1.
2. Create a hash table that holds the count of each **sum[i]** value, where sum[i] = sum(arr[0]+..+arr[i]), for i = 0 to n-1.
3. Now start calculating cumulative sum and then we get increment count by 1 for that sum represented as index in the hash table. Sub-array by each pair of positions with same value of cumulative sum constitute a continuous range with equal number of 1's and 0's.
4. Now traverse the hash table and get the frequency of each element in the hash table. Let frequency be denoted as **freq**. For each **freq** > 1 we can choose any two pair of indices of sub-array by **(freq * (freq − 1)) / 2** number of ways . Do the same for all **freq** and sum up the result that will be the number all possible sub-arrays containing equal number of 1's and 0's.

5. Also add **freq** of the sum 0 in the hash table to the final result.

Explanation:
Considering all 0's as -1, if sum[i] == sum[j], where sum[i] = sum(arr[0]+..+arr[i]) and sum[j] = sum(arr[0]+..+arr[j]) and 'i' is less than 'j', then sum(arr[i+1]+..+arr[j]) must be 0. It can only be 0 if arr(i+1, .., j) contains equal number of 1's and 0's.

C++

```
 // C++ implementation to count subarrays with
// equal number of 1's and 0's
#include <bits/stdc++.h>

using namespace std;

// function to count subarrays with
// equal number of 1's and 0's
int countSubarrWithEqualZeroAndOne(int arr[], int n)
{
    // 'um' implemented as hash table to store
    // frequency of values obtained through
    // cumulative sum
    unordered_map<int, int> um;
    int curr_sum = 0;

    // Traverse original array and compute cumulative
    // sum and increase count by 1 for this sum
    // in 'um'. Adds '-1' when arr[i] == 0
    for (int i = 0; i < n; i++) {
        curr_sum += (arr[i] == 0) ? -1 : arr[i];
        um[curr_sum]++;
    }

    int count = 0;
    // traverse the hash table 'um'
    for (auto itr = um.begin(); itr != um.end(); itr++) {

        // If there are more than one prefix subarrays
        // with a particular sum
        if (itr->second > 1)
            count += ((itr->second * (itr->second - 1)) / 2);
    }

    // add the subarrays starting from 1st element and
    // have equal number of 1's and 0's
    if (um.find(0) != um.end())
        count += um[0];
```

```
    // required count of subarrays
    return count;
}

// Driver program to test above
int main()
{
    int arr[] = { 1, 0, 0, 1, 0, 1, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Count = "
         << countSubarrWithEqualZeroAndOne(arr, n);
    return 0;
}
```

Python3

```
 # Python3 implementation to count
# subarrays with equal number
# of 1's and 0's

# function to count subarrays with
# equal number of 1's and 0's
def countSubarrWithEqualZeroAndOne (arr, n):

    # 'um' implemented as hash table
    # to store frequency of values
    # obtained through cumulative sum
    um = dict()
    curr_sum = 0

    # Traverse original array and compute
    # cumulative sum and increase count
    # by 1 for this sum in 'um'.
    # Adds '-1' when arr[i] == 0
    for i in range(n):
        curr_sum += (-1 if (arr[i] == 0) else arr[i])
        if um.get(curr_sum):
            um[curr_sum]+=1
        else:
            um[curr_sum]=1

    count = 0

    # traverse the hash table 'um'
    for itr in um:

        # If there are more than one
        # prefix subarrays with a
```

```
        # particular sum
        if um[itr] > 1:
            count += ((um[itr] * int(um[itr] - 1)) / 2)

    # add the subarrays starting from
    # 1st element and have equal
    # number of 1's and 0's
    if um.get(0):
        count += um[0]

    # required count of subarrays
    return int(count)

# Driver code to test above
arr = [ 1, 0, 0, 1, 0, 1, 1 ]
n = len(arr)
print("Count =",
    countSubarrWithEqualZeroAndOne(arr, n))

# This code is contributed by "Sharad_Bhardwaj".
```

Output:

```
Count = 8
```

Time Complexity: O(n).
Auxiliary Space: O(n).

Improved By : ravi_Sgo

Source

https://www.geeksforgeeks.org/count-subarrays-equal-number-1s-0s/

Chapter 56

Count subarrays with same even and odd elements

Count subarrays with same even and odd elements - GeeksforGeeks

Given an array of N integers, count number of even-odd subarrays. An even – odd subarray is a subarray that contains the same number of even as well as odd integers.

Examples :

```
Input : arr[] = {2, 5, 7, 8}
Output : 3
Explanation : There are total 3 even-odd subarrays.
               1) {2, 5}
               2) {7, 8}
               3) {2, 5, 7, 8}

Input : arr[] = {3, 4, 6, 8, 1, 10}
Output : 3
Explanation : In this case, 3 even-odd subarrays are:
               1) {3, 4}
               2) {8, 1}
               3) {1, 10}
```

This problem is mainly a variation of count subarrays with equal number of 0s and 1s.

A **naive approach** would be to check for all possible subarrays using two loops, whether they are even-odd subarrays or not. This approach will take $O(N^2)$ time.

An **Efficient approach** solves the problem in O(N) time and it is based on following ideas:

- Even-odd subarrays will always be of even length.

- Maintaining track of the difference between the frequency of even and odd integers.
- Hashing of this difference of frequencies is useful in finding number of even-odd subarrays.

The basic idea is to use the difference between the frequency of odd and even numbers to obtain an optimal solution. We will maintain two integer hash arrays for the positive and negative value of the difference.
-> Example to understand in better way :
-> Consider difference = freq(odd) – freq(even)
-> To calculate this difference, increment the value of 'difference' when there is an odd integer and decrement it when there is an even integer. (initially, difference = 0)
arr[] = {3, 4, 6, 8, 1, 10}

index 0 1 2 3 4 5 6

array 3 4 6 8 1 10

difference 0 1 0 -1 -2 -1 -2

-> Observe that whenever a value 'k' repeats in the 'difference' array, there exists an even-odd subarray for each previous occurrence of that value i.e. subarray exists from index i + 1 to j where difference[i] = k and difference[j] = k.

-> Value '0' is repeated in 'difference' array at index 2 and hence subarray exists for (0, 2] indexes. Similarly, for repetition of values '-1' (at indexes 3 and 5) and '-2' (at indexes 4 and 6), subarray exists for (3, 5] and (4, 6] indexes.

Below is the implementation of the **O(N)** solution described above.

C++

```
 /*C++ program to find total number of
even-odd subarrays present in given array*/
#include <iostream>
using namespace std;

// function that returns the count of subarrays that
// contain equal number of odd as well as even numbers
int countSubarrays(int arr[], int n)
{
    // initialize difference and answer with 0
    int difference = 0;
    int ans = 0;

    // create two auxiliary hash arrays to count frequency
    // of difference, one array for non-negative difference
    // and other array for negative difference. Size of these
    // two auxiliary arrays is 'n+1' because difference can
    // reach maximum value 'n' as well as minimum value '-n'
    int hash_positive[n + 1], hash_negative[n + 1];
```

```
    // initialize these auxiliary arrays with 0
    fill_n(hash_positive, n + 1, 0);
    fill_n(hash_negative, n + 1, 0);

    // since the difference is initially 0, we have to
    // initialize hash_positive[0] with 1
    hash_positive[0] = 1;

    // for loop to iterate through whole
    // array (zero-based indexing is used)
    for (int i = 0; i < n ; i++)
    {
        // incrementing or decrementing difference based on
        // arr[i] being even or odd, check if arr[i] is odd
        if (arr[i] & 1 == 1)
            difference++;
        else
            difference--;

        // adding hash value of 'difference' to our answer
        // as all the previous occurrences of the same
        // difference value will make even-odd subarray
        // ending at index 'i'. After that, we will increment
        // hash array for that 'difference' value for
        // its occurrence at index 'i'. if difference is
        // negative then use hash_negative
        if (difference < 0)
        {
            ans += hash_negative[-difference];
            hash_negative[-difference]++;
        }

        // else use hash_positive
        else
        {
            ans += hash_positive[difference];
            hash_positive[difference]++;
        }
    }

    // return total number of even-odd subarrays
    return ans;
}

// Driver code
int main()
{
```

```
    int arr[] = {3, 4, 6, 8, 1, 10, 5, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Printing total number of even-odd subarrays
    cout << "Total Number of Even-Odd subarrays"
        " are " << countSubarrays(arr,n);

    return 0;
}
```

C#

```
 // C# program to find total
// number of even-odd subarrays
// present in given array
using System;

class GFG
{
    // function that returns the
    // count of subarrays that
    // contain equal number of
    // odd as well as even numbers
    static int countSubarrays(int []arr,
                              int n)
    {
        // initialize difference
        // and answer with 0
        int difference = 0;
        int ans = 0;

        // create two auxiliary hash
        // arrays to count frequency
        // of difference, one array
        // for non-negative difference
        // and other array for negative
        // difference. Size of these
        // two auxiliary arrays is 'n+1'
        // because difference can
        // reach maximum value 'n' as
        // well as minimum value '-n'
        int []hash_positive = new int[n + 1];
        int []hash_negative = new int[n + 1];

        // initialize these
        // auxiliary arrays with 0
        Array.Clear(hash_positive, 0, n + 1);
        Array.Clear(hash_negative, 0, n + 1);
```

```
    // since the difference is
    // initially 0, we have to
    // initialize hash_positive[0] with 1
    hash_positive[0] = 1;

    // for loop to iterate
    // through whole array
    // (zero-based indexing is used)
    for (int i = 0; i < n ; i++)
    {
        // incrementing or decrementing
        // difference based on
        // arr[i] being even or odd,
        // check if arr[i] is odd
        if ((arr[i] & 1) == 1)
            difference++;
        else
            difference--;

        // adding hash value of 'difference'
        // to our answer as all the previous
        // occurrences of the same difference
        // value will make even-odd subarray
        // ending at index 'i'. After that,
        // we will increment hash array for
        // that 'difference' value for its
        // occurrence at index 'i'. if
        // difference is negative then use
        // hash_negative
        if (difference < 0)
        {
            ans += hash_negative[-difference];
            hash_negative[-difference]++;
        }

        // else use hash_positive
        else
        {
            ans += hash_positive[difference];
            hash_positive[difference]++;
        }
    }

    // return total number
    // of even-odd subarrays
    return ans;
}
```

```
    // Driver code
    static void Main()
    {
        int []arr = new int[]{3, 4, 6, 8,
                              1, 10, 5, 7};
        int n = arr.Length;

        // Printing total number
        // of even-odd subarrays
        Console.Write("Total Number of Even-Odd" +
                               " subarrays are " +
                           countSubarrays(arr,n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
Total Number of Even-Odd subarrays are 7
```

Time Complexity : O(N), where N is the number of integers.
Auxiliary Space : O(2N), where N is the number of integers.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/count-subarrays-with-same-even-and-odd-elements/

Chapter 57

Count subsets having distinct even numbers

Count subsets having distinct even numbers - GeeksforGeeks

Given a sequence of n numbers. The task is to count all the subsets of the given set which only have even numbers and all are distinct.
Note: By the property of sets, if two subsets have the same set of elements then they are considered as one. For example: [2, 4, 8] and [4, 2, 8] are considered to be the same.

Examples:

```
Input : {4, 2, 1, 9, 2, 6, 5, 3}
Output : 7
The subsets are:
[4], [2], [6], [4, 2],
[2, 6], [4, 6], [4, 2, 6]

Input : {10, 3, 4, 2, 4, 20, 10, 6, 8, 14, 2, 6, 9}
Output : 127
```

A **simple approach** is to consider all the subsets and check whether they satisfy the given conditions or not. The time complexity will be in exponential.

An **efficient approach** is to count number of distinct even numbers. Let this be **ceven**. And then apply formula:
$\mathbf{2^{ceven} - 1}$
This is similar to counting the number of subsets of a given set of n elements. **1** is subtracted because the null set is not considered.

```
 // C++ implementation to count subsets having
// even numbers only and all are distinct
```

```
#include <bits/stdc++.h>
using namespace std;

// function to count the
// required subsets
int countSubsets(int arr[], int n)
{
    unordered_set<int> us;
    int even_count = 0;

    // inserting even numbers in the set 'us'
    // single copy of each number is retained
    for (int i=0; i<n; i++)
        if (arr[i] % 2 == 0)
            us.insert(arr[i]);

    unordered_set<int>:: iterator itr;

    // counting distinct even numbers
    for (itr=us.begin(); itr!=us.end(); itr++)
        even_count++;

    // total count of required subsets
    return (pow(2, even_count) - 1);
}

// Driver program to test above
int main()
{
    int arr[] = {4, 2, 1, 9, 2, 6, 5, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Number of subsets = "
         << countSubsets(arr, n);
    return 0;
}
```

Output:

```
Number of subsets = 7
```

Time Complexity: O(n)

Improved By : neriomo

Source

https://www.geeksforgeeks.org/count-subsets-distinct-even-numbers/

Chapter 58

Count the number of subarrays having a given XOR

Count the number of subarrays having a given XOR - GeeksforGeeks

Given an array of integers arr[] and a number m, count the number of subarrays having XOR of their elements as m.

Examples:

```
Input : arr[] = {4, 2, 2, 6, 4}, m = 6
Output : 4
Explanation : The subarrays having XOR of
              their elements as 6 are {4, 2},
              {4, 2, 2, 6, 4}, {2, 2, 6},
               and {6}

Input : arr[] = {5, 6, 7, 8, 9}, m = 5
Output : 2
Explanation : The subarrays having XOR of
              their elements as 2 are {5}
              and {5, 6, 7, 8, 9}
```

A **Simple Solution** is to use two loops to go through all possible subarrays of arr[] and count the number of subarrays having XOR of their elements as m.

C++

```
 // A simple C++ Program to count all subarrays having
// XOR of elements as given value m
#include <bits/stdc++.h>
using namespace std;
```

```
// Simple function that returns count of subarrays
// of arr with XOR value equals to m
long long subarrayXor(int arr[], int n, int m)
{
    long long ans = 0; // Initialize ans

    // Pick starting point i of subarrays
    for (int i = 0; i < n; i++) {
        int xorSum = 0; // Store XOR of current subarray

        // Pick ending point j of subarray for each i
        for (int j = i; j < n; j++) {
            // calculate xorSum
            xorSum = xorSum ^ arr[j];

            // If xorSum is equal to given value,
            // increase ans by 1.
            if (xorSum == m)
                ans++;
        }
    }
    return ans;
}

// Driver program to test above function
int main()
{
    int arr[] = { 4, 2, 2, 6, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 6;

    cout << "Number of subarrays having given XOR is "
         << subarrayXor(arr, n, m);
    return 0;
}
```

Java

```
 // A simple Java Program to count all
// subarrays having XOR of elements
// as given value m
public class GFG {

    // Simple function that returns
    // count of subarrays of arr with
    // XOR value equals to m
    static long subarrayXor(int arr[],
```

```
                            int n, int m)
    {

        // Initialize ans
        long ans = 0;

        // Pick starting point i of
        // subarrays
        for (int i = 0; i < n; i++)
        {

            // Store XOR of current
            // subarray
            int xorSum = 0;

            // Pick ending point j of
            // subarray for each i
            for (int j = i; j < n; j++)
            {

                // calculate xorSum
                xorSum = xorSum ^ arr[j];

                // If xorSum is equal to
                // given value, increase
                // ans by 1.
                if (xorSum == m)
                    ans++;
            }
        }

        return ans;
    }

    // Driver code
    public static void main(String args[])
    {

        int[] arr = { 4, 2, 2, 6, 4 };
        int n = arr.length;
        int m = 6;

        System.out.println("Number of subarrays"
                           + " having given XOR is "
                           + subarrayXor(arr, n, m));
    }
}
```

```
// This code is contributed by Sam007.
```

C#

```
 // A simple C# Program to count all
// subarrays having XOR of elements
// as given value m
using System;

class GFG {

    // Simple function that returns
    // count of subarrays of arr
    // with XOR value equals to m
    static long subarrayXor(int[] arr,
                            int n, int m)
    {

        // Initialize ans
        long ans = 0;

        // Pick starting point i of
        // subarrays
        for (int i = 0; i < n; i++)
        {

            // Store XOR of current
            // subarray
            int xorSum = 0;

            // Pick ending point j of
            // subarray for each i
            for (int j = i; j < n; j++)
            {

                // calculate xorSum
                xorSum = xorSum ^ arr[j];

                // If xorSum is equal to
                // given value, increase
                // ans by 1.
                if (xorSum == m)
                    ans++;
            }
        }

        return ans;
    }
```

```
    // Driver Program
    public static void Main()
    {
        int[] arr = { 4, 2, 2, 6, 4 };
        int n = arr.Length;
        int m = 6;

        Console.Write("Number of subarrays"
                  + " having given XOR is "
                  + subarrayXor(arr, n, m));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// A simple PHP Program to
// count all subarrays having
// XOR of elements as given value m

// Simple function that returns
// count of subarrays of arr
// with XOR value equals to m
function subarrayXor($arr, $n,$m)
{

    // Initialize ans
    $ans = 0;

    // Pick starting point
    // i of subarrays
    for ($i = 0; $i < $n; $i++)
    {

        // Store XOR of
        // current subarray
        $xorSum = 0;

        // Pick ending point j of
        // subarray for each i
        for ($j = $i; $j < $n; $j++)
        {
            // calculate xorSum
            $xorSum = $xorSum ^ $arr[$j];
```

```
            // If xorSum is equal
            // to given value,
            // increase ans by 1.
            if ($xorSum == $m)
                $ans++;
        }
    }
    return $ans;
}

    // Driver Code
    $arr = array(4, 2, 2, 6, 4);
    $n = count($arr);
    $m = 6;

    echo "Number of subarrays having given XOR is "
         , subarrayXor($arr, $n, $m);

// This code is contributed by anuj_67.
?>
```

Output:

```
Number of subarrays having given XOR is 4
```

Time Complexity of above solution is $O(n^2)$.

An **Efficient Solution** solves the above problem in O(n) time. Let us call the XOR of all elements in the range [i+1, j] as A, in the range [0, i] as B, and in the range [0, j] as C. If we do XOR of B with C, the overlapping elements in [0, i] from B and C zero out and we get XOR of all elements in the range [i+1, j], i.e. A. Since A = B XOR C, we have B = A XOR C. Now, if we know the value of C and we take the value of A as m, we get the count of A as the count of all B satisfying this relation. Essentially, we get the count of all subarrays having XOR-sum m for each C. As we take sum of this count over all C, we get our answer.

```
1) Initialize ans as 0.
2) Compute xorArr, the prefix xor-sum array.
3) Create a map mp in which we store count of
   all prefixes with XOR as a particular value.
4) Traverse xorArr and for each element in xorArr
   (A) If m^xorArr[i] XOR exists in map, then
       there is another previous prefix with
       same XOR, i.e., there is a subarray ending
       at i with XOR equal to m. We add count of
       all such subarrays to result.
   (B) If xorArr[i] is equal to m, increment ans by 1.
```

```
   (C) Increment count of elements having XOR-sum
       xorArr[i] in map by 1.
5) Return ans.
```

```
 // C++ Program to count all subarrays having
// XOR of elements as given value m with
// O(n) time complexity.
#include <bits/stdc++.h>
using namespace std;

// Returns count of subarrays of arr with XOR
// value equals to m
long long subarrayXor(int arr[], int n, int m)
{
    long long ans = 0; // Initialize answer to be returned

    // Create a prefix xor-sum array such that
    // xorArr[i] has value equal to XOR
    // of all elements in arr[0 ..... i]
    int* xorArr = new int[n];

    // Create map that stores number of prefix array
    // elements corresponding to a XOR value
    unordered_map<int, int> mp;

    // Initialize first element of prefix array
    xorArr[0] = arr[0];

    // Computing the prefix array.
    for (int i = 1; i < n; i++)
        xorArr[i] = xorArr[i - 1] ^ arr[i];

    // Calculate the answer
    for (int i = 0; i < n; i++) {
        // Find XOR of current prefix with m.
        int tmp = m ^ xorArr[i];

        // If above XOR exists in map, then there
        // is another previous prefix with same
        // XOR, i.e., there is a subarray ending
        // at i with XOR equal to m.
        ans = ans + ((long long)mp[tmp]);

        // If this subarray has XOR equal to m itself.
        if (xorArr[i] == m)
            ans++;

        // Add the XOR of this subarray to the map
```

```
        mp[xorArr[i]]++;
    }

    // Return total count of subarrays having XOR of
    // elements as given value m
    return ans;
}

// Driver program to test above function
int main()
{
    int arr[] = { 4, 2, 2, 6, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 6;

    cout << "Number of subarrays having given XOR is "
         << subarrayXor(arr, n, m);
    return 0;
}
```

Output:

```
Number of subarrays having given XOR is 4
```

Time Complexity: O(n)

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/count-number-subarrays-given-xor/

Chapter 59

Counting frequencies of array elements

Counting frequencies of array elements - GeeksforGeeks

Given an array which may contain duplicates, print all elements and their frequencies.

Examples:

```
Input :  arr[] = {10, 20, 20, 10, 10, 20, 5, 20}
Output : 10 3
         20 4
         5  1

Input : arr[] = {10, 20, 20}
Output : 10 2
         20 1
```

A **simple solution** is to run two loops. For every item count number of times it occurs. To avoid duplicate printing, keep track of processed items.

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n)
{
    // Mark all array elements as not visited
    vector<bool> visited(n, false);

    // Traverse through array elements and
    // count frequencies
```

```
    for (int i = 0; i < n; i++) {

        // Skip this element if already processed
        if (visited[i] == true)
            continue;

        // Count frequency
        int count = 1;
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                visited[j] = true;
                count++;
            }
        }
        cout << arr[i] << " " << count << endl;
    }
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
10 3
20 4
5 1
```

Time Complexity : O(n^2)
Auxiliary Space : O(n)

An **efficient solution** is to use hashing.

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
```

```
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Traverse through map and print frequencies
    for (auto x : mp)
        cout << x.first << " " << x.second << endl;
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
5 1
10 3
20 4
```

Time Complexity : O(n)
Auxiliary Space : O(n)

In above efficient solution, how to print elements in same order as they appear in input?

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // To print elements according to first
    // occurrence, traverse array one more time
    // print frequencies of elements and mark
    // frequencies as -1 so that same element
    // is not printed multiple times.
    for (int i = 0; i < n; i++) {
```

```
      if (mp[arr[i]] != -1)
      {
          cout << arr[i] << " " << mp[arr[i]] << endl;
          mp[arr[i]] = -1;
      }
    }
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
10 3
20 4
5 1
```

Time Complexity : O(n)
Auxiliary Space : O(n)

This problem can be solved in Java using Hashmap. Below is the program.

```
 // Java prorgam to count frequencies of
// integers in array using Hashmap
import java.io.*;
import java.util.*;
class OccurenceOfNumberInArray {
    static void frequencyNumber(int arr[], int size)
    {
        // Creating a HashMap containing integer
        // as a key and occurrences as a value
        HashMap<Integer, Integer> freqMap
            = new HashMap<Integer, Integer>();

        for (int i=0;i<size;i++) {
            if (freqMap.containsKey(arr[i])) {

                // If number is present in freqMap,
                // incrementing it's count by 1
                freqMap.put(arr[i], freqMap.get(arr[i]) + 1);
            }
            else {
```

```
                // If integer is not present in freqMap,
                // putting this integer to freqMap with 1 as it's value
                freqMap.put(arr[i], 1);
            }
        }

        // Printing the freqMap
        for (Map.Entry entry : freqMap.entrySet()) {
            System.out.println(entry.getKey() + " " + entry.getValue());
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        int arr[] = {10, 20, 20, 10, 10, 20, 5, 20};
        int size = arr.length;
        frequencyNumber(arr,size);
    }
}
```

Improved By : Ajit kumar panigrahy

Source

https://www.geeksforgeeks.org/counting-frequencies-of-array-elements/

Chapter 60

Cuckoo Hashing – Worst case O(1) Lookup!

Cuckoo Hashing - Worst case O(1) Lookup! - GeeksforGeeks

Background :

There are three basic operations that must be supported by a hash table (or a dictionary):

- **Lookup(key):** return **true** if key is there in the table, else **false**
- **Insert(key):** adds the item 'key' to the table if not already present
- **Delete(key):** removes 'key' from the table

Collisions are very likely even if we have a big table to store keys. Using the results from the birthday paradox: with only 23 persons, the probability that two people share the same birth date is 50%! There are 3 general strategies towards resolving hash collisions:

- **Closed addressing or Chaining**: store colliding elements in an auxiliary data structure like a linked list or a binary search tree.
- **Open addressing**: allow elements to overflow out of their target bucket and into other spaces.

Although above solutions provide expected lookup cost as O(1), the expected worst-case cost of a lookup in Open Addressing (with linear probing) is $\Omega(\log n)$ and $\Theta(\log n / \log \log n)$ in simple chaining (Source : Standford Lecture Notes). To close the gap of expected time and worst case expected time, two ideas are used:

- **Multiple-choice hashing**: Give each element multiple choices for positions where it can reside in the hash table
- **Relocation hashing**: Allow elements in the hash table to move after being placed

Cuckoo Hashing :

Cuckoo hashing applies the idea of multiple-choice and relocation together and guarantees O(1) worst case lookup time!

- **Multiple-choice:** We give a key **two choices** h1(key) and h2(key) for residing.
- **Relocation**: It may happen that h1(key) and h2(key) are preoccupied. This is resolved by imitating the Cuckoo bird: *it pushes the other eggs or young out of the nest when it hatches*. Analogously, inserting a new key into a cuckoo hashing table may push an older key to a different location. This leaves us with the problem of re-placing the older key.
  - If alternate position of older key is vacant, there is no problem.
  - Otherwise, older key displaces another key. This continues until the procedure finds a vacant position, or enters a cycle. In case of cycle, new hash functions are chosen and the whole data structure is 'rehashed'. Multiple rehashes might be necessary before Cuckoo succeeds.

Insertion is expected O(1) (amortized) with high probability, even considering the possibility rehashing, as long as the number of keys is kept below half of the capacity of the hash table, i.e., the load factor is below 50%.

Deletion is O(1) worst-case as it requires inspection of just two locations in the hash table.

Illustration :

Input:

```
{20, 50, 53, 75, 100, 67, 105, 3, 36, 39}
```

Hash Functions:

```
h1(key) = key%11
h2(key) = (key/11)%11
```

	20	50	53	75	100	67	105	3	36	39
h_1(key)	9	6	9	9	1	1	6	3	3	6
h_2(key)	1	4	4	6	9	6	9	0	3	3

Let's start with inserting **20** at its possible position in the first table determined by h1(20):

table[1]	-	-	-	-	-	-	-	-	-	20	-
table[2]	-	-	-	-	-	-	-	-	-	-	-

Nex

table[1]	-	-	-	-	-	-	50	-	-	20	-
table[2]	-	-	-	-	-	-	-	-	-	-	-

50

Next: **53**. h1(53) = 9. But 20 is already there at 9. We place 53 in table 1 & 20 in table 2 at h2(20)

.

table[1]	-	-	-	-	-	-	50	-	-	53	-
table[2]	-	20	-	-	-	-	-	-	-	-	-

Next: **75**. h1(75) = 9. But 53 is already there at 9. We place 75 in table 1 & 53 in table 2 at h2(53)

table[1]	-	-	-	-	-	-	50	-	-	75	-
table[2]	-	20	-	-	53	-	-	-	-	-	-

Next: **100**. h1(100) = 1.

table[1]	-	100	-	-	-	-	50	-	-	75	-
table[2]	-	20	-	-	53	-	-	-	-	-	-

Next: **67**. h1(67) = 1. But 100 is already there at 1. We place 67 in table 1 & 100 in table 2

table[1]	-	67	-	-	-	-	50	-	-	75	-
table[2]	-	20	-	-	53	-	-	-	-	100	-

Next: **105**. h1(105) = 6. But 50 is already there at 6. We place 105 in table 1 & 50 in table 2 at h2(50) = 4. Now 53 has been displaced. h1(53) = 9. 75 displaced: h2(75) = 6.

table[1]	-	67	-	-	-	-	105	-	-	53	-
table[2]	-	20	-	-	50	-	75	-	-	100	-

Next: **3**. h1(3) = 3.

table[1]	-	67	-	3	-	-	105	-	-	53	-
table[2]	-	20	-	-	50	-	75	-	-	100	-

Next: **36**. h1(36) = 3. h2(3) = 0.

table[1]	-	67	-	36	-	-	105	-	-	53	-
table[2]	3	20	-	-	50	-	75	-	-	100	-

Ne

39. h1(39) = 6. h2(105) = 9. h1(100) = 1. h2(67) = 6. h1(75) = 9. h2(53) = 4. h1(50) = 6. h2(39) = 3.

Here, the new key 39 is displaced later in the recursive calls to place 105 which it displaced.

table[1]	-	100	-	36	-	-	50	-	-	75	-
table[2]	3	20	-	39	53	-	67	-	-	105	-

Implementation :

Below is C/C++ implementation of Cuckoo hashing

```
 // C++ program to demonstrate working of Cuckoo
// hashing.
#include<bits/stdc++.h>

// upper bound on number of elements in our set
```

```
#define MAXN 11

// choices for position
#define ver 2

// Auxiliary space bounded by a small multiple
// of MAXN, minimizing wastage
int hashtable[ver][MAXN];

// Array to store possible positions for a key
int pos[ver];

/* function to fill hash table with dummy value
 * dummy value: INT_MIN
 * number of hashtables: ver */
void initTable()
{
    for (int j=0; j<MAXN; j++)
        for (int i=0; i<ver; i++)
            hashtable[i][j] = INT_MIN;
}

/* return hashed value for a key
 * function: ID of hash function according to which
    key has to hashed
 * key: item to be hashed */
int hash(int function, int key)
{
    switch (function)
    {
        case 1: return key%MAXN;
        case 2: return (key/MAXN)%MAXN;
    }
}

/* function to place a key in one of its possible positions
 * tableID: table in which key has to be placed, also equal
   to function according to which key must be hashed
 * cnt: number of times function has already been called
   in order to place the first input key
 * n: maximum number of times function can be recursively
   called before stopping and declaring presence of cycle */
void place(int key, int tableID, int cnt, int n)
{
    /* if function has been recursively called max number
       of times, stop and declare cycle. Rehash. */
    if (cnt==n)
    {
```

```
        printf("%d unpositioned\n", key);
        printf("Cycle present. REHASH.\n");
        return;
    }

    /* calculate and store possible positions for the key.
     * check if key already present at any of the positions.
       If YES, return. */
    for (int i=0; i<ver; i++)
    {
        pos[i] = hash(i+1, key);
        if (hashtable[i][pos[i]] == key)
           return;
    }

    /* check if another key is already present at the
       position for the new key in the table
     * If YES: place the new key in its position
     * and place the older key in an alternate position
       for it in the next table */
    if (hashtable[tableID][pos[tableID]]!=INT_MIN)
    {
        int dis = hashtable[tableID][pos[tableID]];
        hashtable[tableID][pos[tableID]] = key;
        place(dis, (tableID+1)%ver, cnt+1, n);
    }
    else //else: place the new key in its position
       hashtable[tableID][pos[tableID]] = key;
}

/* function to print hash table contents */
void printTable()
{
    printf("Final hash tables:\n");

    for (int i=0; i<ver; i++, printf("\n"))
        for (int j=0; j<MAXN; j++)
            (hashtable[i][j]==INT_MIN)? printf("- "):
                     printf("%d ", hashtable[i][j]);

    printf("\n");
}

/* function for Cuckoo-hashing keys
 * keys[]: input array of keys
 * n: size of input array */
void cuckoo(int keys[], int n)
{
```

```
    // initialize hash tables to a dummy value (INT-MIN)
    // indicating empty position
    initTable();

    // start with placing every key at its position in
    // the first hash table according to first hash
    // function
    for (int i=0, cnt=0; i<n; i++, cnt=0)
        place(keys[i], 0, cnt, n);

    //print the final hash tables
    printTable();
}

/* driver function */
int main()
{
    /* following array doesn't have any cycles and
       hence  all keys will be inserted without any
       rehashing */
    int keys_1[] = {20, 50, 53, 75, 100, 67, 105,
                    3, 36, 39};

    int n = sizeof(keys_1)/sizeof(int);

    cuckoo(keys_1, n);

    /* following array has a cycle and hence we will
       have to rehash to position every key */
    int keys_2[] = {20, 50, 53, 75, 100, 67, 105,
                    3, 36, 39, 6};

    int m = sizeof(keys_2)/sizeof(int);

    cuckoo(keys_2, m);

    return 0;
}
```

Output:

```
Final hash tables:
- 100 - 36 - - 50 - - 75 -
3 20 - 39 53 - 67 - - 105 -

105 unpositioned
Cycle present. REHASH.
Final hash tables:
```

```
- 67 - 3 - - 39 - - 53 -
6 20 - 36 50 - 75 - - 100 -
```

Generalizations of cuckoo hashing that use more than 2 alternative hash functions can be expected to utilize a larger part of the capacity of the hash table efficiently while sacrificing some lookup and insertion speed. Example: if we use 3 hash functions, it's safe to load 91% and still be operating within expected bounds (Source : Wiki)

Source

https://www.geeksforgeeks.org/cuckoo-hashing/

Chapter 61

Cumulative frequency of count of each element in an unsorted array

Cumulative frequency of count of each element in an unsorted array - GeeksforGeeks

Given an unsorted array. The task is to calculate the **cumulative frequency** of each element of the array using count array.

Examples:

```
Input : arr[] = [1, 2, 2, 1, 3, 4]
Output :1->2
        2->4
        3->5
        4->6

Input : arr[] = [1, 1, 1, 2, 2, 2]
Output :1->3
        2->6
```

A **simple solution** is to use two nested loops, the outer loops picks an element from left to right that are not visited. The inner loop counts its occurrences and mark occurrences as visited. Time complexity of this solution is O(n*n) and auxiliary space required is O(n).

A **better solution** is to use sorting. We sort the array so that same elements come together. After sorting, we linearly traverse elements and count their frequencies.

An **efficient solution** is to use hashing. Insert the element and its frequency in a set of pairs. As the set stores unique values in a sorted order, it will store all the elements with their frequencies in a sorted order. Iterate in the set and print the frequencies by adding the previous ones at every step.

Below is the implementation of the above approach.

```
 // CPP program to count cumlative
// frequencies of elements in an unsorted array.
#include <bits/stdc++.h>
using namespace std;

void countFreq(int a[], int n)
{
    // Insert elements and their
    // frequencies in hash map.
    unordered_map<int, int> hm;
    for (int i = 0; i < n; i++)
        hm[a[i]]++;

    // Declare a set
    set<pair<int, int> > st;

    // insert the element and
    // and insert its frequency in a set
    for (auto x : hm) {
        st.insert({ x.first, x.second });
    }

    int cumul = 0;

    // iterate the set and print the
    // cumulative frequency
    for (auto x : st) {
        cumul += x.second;
        cout << x.first << " " << cumul << endl;
    }
}

// Driver Code
int main()
{
    int a[] = { 1, 3, 2, 4, 2, 1 };
    int n = sizeof(a) / sizeof(a[0]);
    countFreq(a, n);
    return 0;
}
```

Output:

```
1 2
```

```
2 4
3 5
4 6
```

Time complexity of the solution is **O(n log n)**.

What if we need frequencies of elements according to the order of the first occurrence?
For example, an array [2, 4, 1, 2, 1, 3, 4], the frequency of 2 should be printed first, then of 4, then 1 and finally 3.
Approach: Hash the count of occurrences of an element. Traverse in the array and print the cumulative frequency. Once the element and its cumulative frequency has been printed, hash the occurrence of that element as 0 so that it not printed again if it appears in the latter half of array while traversal.

Below is the implementation of the above approach:

```
 // CPP program to print the cumulative frequency
// according to the order given
#include <bits/stdc++.h>
using namespace std;

// Function to print the cumulative frequency
// according to the order given
void countFreq(int a[], int n)
{
    // Insert elements and their
    // frequencies in hash map.
    unordered_map<int, int> hm;
    for (int i=0; i<n; i++)
        hm[a[i]]++;
    int cumul = 0;

   // traverse in the array
   for(int i=0;i<n;i++)
   {
       // add the frequencies
       cumul += hm[a[i]];

       // if the element has not been
       // visited previously
       if(hm[a[i]])
       {
           cout << a[i] << "->" << cumul << endl;
       }
       // mark the hash 0
       // as the element's cumulative frequency
       // has been printed
       hm[a[i]]=0;
```

```
   }
}

// Driver Code
int main()
{
    int a[] = {1, 3, 2, 4, 2, 1};
    int n = sizeof(a)/sizeof(a[0]);
    countFreq(a, n);
    return 0;
}
```

Output:

```
1->2
3->3
2->5
4->6
```

Source

https://www.geeksforgeeks.org/cumulative-frequency-of-count-of-each-element-in-an-unsorted-array/

Chapter 62

DBMS – File Organization | Set 4

DBMS - File Organization | Set 4 - GeeksforGeeks

Prerequisite – Hashing Data Structure
In database management system, When we want to retrieve a particular data, It becomes very inefficient to search all the index values and reach the desired data. In this situation, Hashing technique comes into picture.
Hashing is an efficient technique to directly search the location of desired data on the disk without using index structure. Data is stored at the data blocks whose address is generated by using hash function. The memory location where these records are stored is called as data block or data bucket.

- **Data bucket** – Data buckets are the memory locations where the records are stored. These buckets are also considered as *Unit Of Storage*.
- **Hash Function** – Hash function is a mapping function that maps all the set of search keys to actual record address. Generally, hash function uses primary key to generate the hash index – address of the data block. Hash function can be simple mathematical function to any complex mathematical function.
- **Hash Index-**The prefix of an entire hash value is taken as a hash index. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2n buckets. When all these bits are consumed ? then the depth value is increased linearly and twice the buckets are allocated.

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if we want to generate address for STUDENT_ID = 76 using mod (5) hash function, it always result in the same bucket address 4. There will not be any changes to the bucket address here. Hence number of data buckets in the memory for this static hashing remains constant throughout.

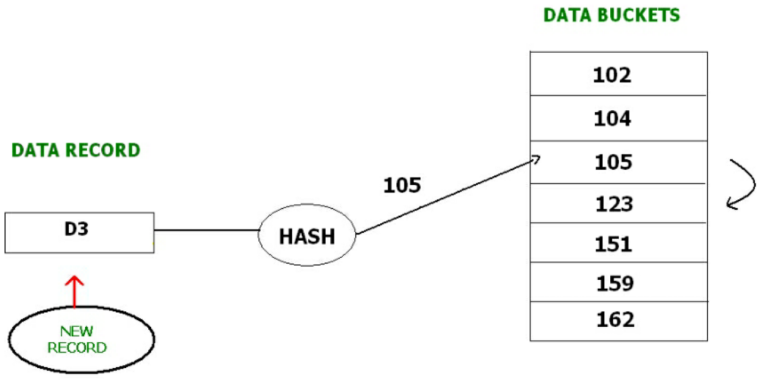
Operations –

- **Insertion** – When a new record is inserted into the table, The hash function h generate a bucket address for the new record based on its hash key K.
  Bucket address = h(K)
- **Searching** – When a record needs to be searched, The same hash function is used to retrieve the bucket address for the record. For Example, if we want to retrieve whole record for ID 76, and if the hash function is mod (5) on that ID, the bucket address generated would be 4. Then we will directly got to address 4 and retrieve the whole record for ID 104. Here ID acts as a hash key.
- **Deletion** – If we want to delete a record, Using the hash function we will first fetch the record which is supposed to be deleted. Then we will remove the records for that address in memory.
- **Updation** – The data record that needs to be updated is first searched using hash function, and then the data record is updated.

Now, If we want to insert some new records into the file But the data bucket address generated by the hash function is not empty or the data already exists in that address. This becomes a critical situation to handle. This situation in the static hashing is called **bucket overflow**.
How will we insert data in this case?
There are several methods provided to overcome this situation. Some commonly used methods are discussed below:

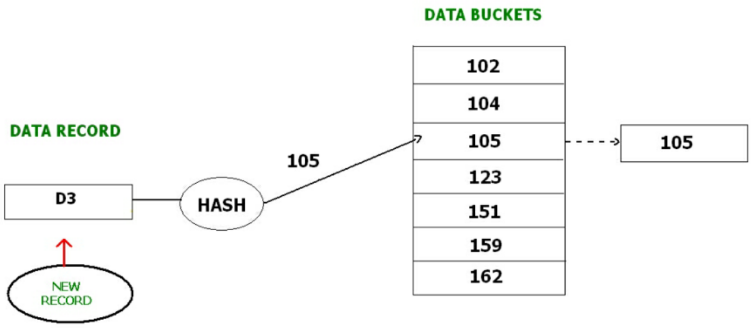
1. **Open Hashing** –
   In Open hashing method, next available data block is used to enter the new record, instead of overwriting older one. This method is also called linear probing.

   For example, D3 is a new record which needs to be inserted , the hash function generates address as 105. But it is already full. So the system searches next available data bucket, 123 and assigns D3 to it.



2. **Closed hashing** –
   In Closed hashing method, a new data bucket is allocated with same address and is linked it after the full data bucket. This method is also known as overflow chaining. For example, we have to insert a new record D3 into the tables. The static hash function generates the data bucket address as 105. But this bucket is full to store the

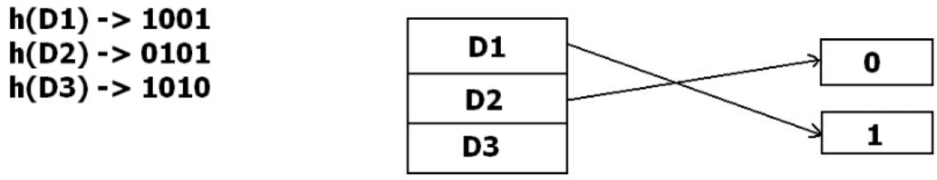
new data. In this case is a new data bucket is added at the end of 105 data bucket and is linked to it. Then new record D3 is inserted into the new bucket.



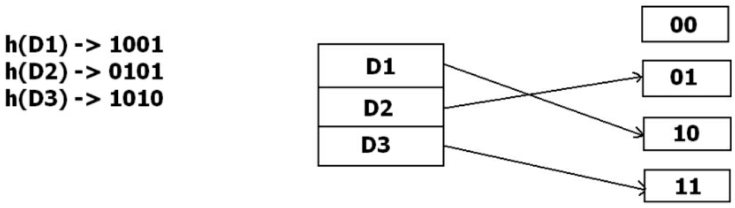
- **Quadratic probing :**
  Quadratic probing is very much similar to open hashing or linear probing. Here, The only difference between old and new bucket is linear. Quadratic function is used to determine the new bucket address.
- **Double Hashing :**
  Double Hashing is another method similar to linear probing. Here the difference is fixed as in linear probing, but this fixed difference is calculated by using another hash function. That's why the name is double hashing.

The drawback of static hashing is that that it does not expand or shrink dynamically as the size of the database grows or shrinks. In Dynamic hashing, data buckets grows or shrinks (added or removed dynamically) as the records increases or decreases. Dynamic hashing is also known as extended hashing.

In dynamic hashing, the hash function is made to produce a large number of values. For Example, there are three data records D1, D2 and D3 . The hash function generates three addresses 1001, 0101 and 1010 respectively. This method of storing considers only part of this address – especially only first one bit to store the data. So it tries to load three of them at address 0 and 1.



But the problem is that No bucket address is remaining for D3. The bucket has to grow dynamically to accommodate D3. So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2 bit address. Then it tries to accommodate D3.



Reference –
cse.iitb.ac.in

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/hashing-in-dbms/

Chapter 63

Design a data structure that supports insert, delete, search and getRandom in constant time

Design a data structure that supports insert, delete, search and getRandom in constant time - GeeksforGeeks

Design a data structure that supports following operations in $\Theta(1)$ time.

insert(x): Inserts an item x to the data structure if not already present.

remove(x): Removes an item x from the data structure if present.

search(x): Searches an item x in the data structure.

getRandom(): Returns a random element from current set of elements

We can use hashingto support first 3 operations in $\Theta(1)$ time. How to do the 4th operation? The idea is to use a resizable array (ArrayList in Java, vector in C) together with hashing. Resizable arrays support insert in $\Theta(1)$ amortized time complexity. To implement getRandom(), we can simply pick a random number from 0 to size-1 (size is number of current elements) and return the element at that index. The hash map stores array values as keys and array indexes as values.

Following are detailed operations.

insert(x)
1) Check if x is already present by doing a hash map lookup.
2) If not present, then insert it at the end of the array.
3) Add in hash table also, x is added as key and last array index as index.

remove(x)
1) Check if x is present by doing a hash map lookup.

2) If present, then find its index and remove it from hash map.
3) Swap the last element with this element in array and remove the last element.
Swapping is done because the last element can be removed in O(1) time.
4) Update index of last element in hash map.

getRandom()
1) Generate a random number from 0 to last index.
2) Return the array element at the randomly generated index.

search(x)
Do a lookup for x in hash map.

Below is implementation of the data structure.

Java

```
 /* Java program to design a data structure that support folloiwng operations
   in Theta(n) time
   a) Insert
   b) Delete
   c) Search
   d) getRandom */
import java.util.*;

// class to represent the required data structure
class MyDS
{
   ArrayList<Integer> arr;   // A resizable array

   // A hash where keys are array elements and vlaues are
   // indexes in arr[]
   HashMap<Integer, Integer>  hash;

   // Constructor (creates arr[] and hash)
   public MyDS()
   {
       arr = new ArrayList<Integer>();
       hash = new HashMap<Integer, Integer>();
   }

   // A Theta(1) function to add an element to MyDS
   // data structure
   void add(int x)
   {
      // If ekement is already present, then noting to do
      if (hash.get(x) != null)
          return;

      // Else put element at the end of arr[]
```

```
    int s = arr.size();
    arr.add(x);

    // And put in hash also
    hash.put(x, s);
}

// A Theta(1) function to remove an element from MyDS
// data structure
void remove(int x)
{
    // Check if element is present
    Integer index = hash.get(x);
    if (index == null)
       return;

    // If present, then remove element from hash
    hash.remove(x);

    // Swap element with last element so that remove from
    // arr[] can be done in O(1) time
    int size = arr.size();
    Integer last = arr.get(size-1);
    Collections.swap(arr, index,  size-1);

    // Remove last element (This is O(1))
    arr.remove(size-1);

    // Update hash table for new index of last element
    hash.put(last, index);
}

// Returns a random element from MyDS
int getRandom()
{
    // Find a random index from 0 to size - 1
    Random rand = new Random();  // Choose a different seed
    int index = rand.nextInt(arr.size());

    // Return element at randomly picked index
    return arr.get(index);
}

// Returns index of element if element is present, otherwise null
Integer search(int x)
{
    return hash.get(x);
}
```

```
}

// Driver class
class Main
{
    public static void main (String[] args)
    {
        MyDS ds = new MyDS();
        ds.add(10);
        ds.add(20);
        ds.add(30);
        ds.add(40);
        System.out.println(ds.search(30));
        ds.remove(20);
        ds.add(50);
        System.out.println(ds.search(50));
        System.out.println(ds.getRandom());
    }
}
```

C++

```
 /* C++ program to design a DS that supports folloiwng operations
in Theta(n) time
a) Insert
b) Delete
c) Search
d) getRandom */

#include<bits/stdc++.h>
using namespace std;

// class to represent the required data structure
class myStructure
{
    // A resizable array
    vector <int> arr;

    // A hash where keys are array elements and vlaues are
    // indexes in arr[]
    map <int, int> Map;

    public:
    // A Theta(1) function to add an element to MyDS
    // data structure
    void add(int x)
    {
        // If ekement is already present, then noting to do
```

```
    if(Map.find(x) != Map.end())
        return;

    // Else put element at the end of arr[]
    int index = arr.size();
    arr.push_back(x);

    // and hashmap also
    Map.insert(std::pair<int,int>(x, index));
}

// function to remove a number to DS in O(1)
void remove(int x)
{
    // element not found then return
    if(Map.find(x) == Map.end())
        return;

    // remove element from map
    int index = Map.at(x);
    Map.erase(x);

    // swap with last element in arr
    // then remove element at back
    int last = arr.size() - 1;
    swap(arr[index], arr[last]);
    arr.pop_back();

    // Update hash table for new index of last element
    Map.at(arr[index]) = index;
}

// Returns index of element if element is present, otherwise null
int search(int x)
{
    if(Map.find(x) != Map.end())
    return Map.at(x);
    return -1;
}

// Returns a random element from myStructure
int getRandom()
{
    // Find a random index from 0 to size - 1
    srand (time(NULL));
    int random_index = rand() % arr.size();

    // Return element at randomly picked index
```

```
        return arr.at(random_index);
    }
};

// Driver main
int main()
{
    myStructure ds;
    ds.add(10);
    ds.add(20);
    ds.add(30);
    ds.add(40);
    cout << ds.search(30) << endl;
    ds.remove(20);
    ds.add(50);
    cout << ds.search(50) << endl;
    cout << ds.getRandom() << endl;
}

// This code is contributed by Aditi Sharma
```

Output:

```
2
3
40
```

This article is contributed by **Manish Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : SwapnilShukla1

Source

https://www.geeksforgeeks.org/design-a-data-structure-that-supports-insert-delete-search-and-getrandom-in-consta

Chapter 64

Difference between Inverted Index and Forward Index

Difference between Inverted Index and Forward Index - GeeksforGeeks

Inverted Index

1. It is a data structure that stores mapping from words to documents or set of documents i.e. directs you from word to document.
2. Steps to build Inverted index are:
   - Fetch the document and gather all the words.
   - Check for each word, if it is present then add reference of document to index else create new entry in index for that word.
   - Repeat above steps for all documents and sort the words.
3. Indexing is slow as it first checks that word is present or not.
4. Searching is very fast.
5. Example of Inverted index:

Word	Documents
hello	doc1
sky	doc1, doc3
coffee	doc2
hi	doc2
greetings	doc3

   It does not store duplicate keywords in index.
6. Real life examples of Inverted index:

- Index at the back of the book.
- Reverse lookup

Forward Index:

1. It is a data structure that stores mapping from documents to words i.e. directs you from document to word.
2. Steps to build Forward index are:
   - Fetch the document and gather all the keywords.
   - Append all the keywords in the index entry for this document.
   - Repeat above steps for all documents
3. Indexing is quite fast as it only append keywords as it move forwards.
4. Searching is quite difficult as it has to look at every contents of index just to retrieve all pages related to word.
5. Example of forward index:

Document	Keywords
doc1	hello, sky, morning
doc2	tea, coffee, hi
doc3	greetings, sky

   It stores duplicate keywords in index. Eg: word "sky" is stored multiple times.
6. Real life examples of Forward index:
   - Table of contents in book.
   - DNS lookup

Similarity between Forward index and Inverted Index:

- Both are used to search text in document or set of documents.

Source

https://www.geeksforgeeks.org/difference-inverted-index-forward-index/

Chapter 65

Difference between highest and least frequencies in an array

Difference between highest and least frequencies in an array - GeeksforGeeks

Given an array, find the difference between highest occurrence and least occurrence of any number in an array

Examples:

```
Input  : arr[] = [7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5]
Output : 2
Lowest occurring element (5) occurs once.
Highest occurring element (1 or 7) occurs 3 times

Input  : arr[] = [1, 1, 1, 3, 3, 3]
Output : 0
```

A **simple solution** is to use two loops to count frequency of every element and keep track of maximum and minimum frequencies.

A **better solution** is to sort the array in **O(n log n)** and check
consecutive element's occurrence and compare their count respectively.

CPP

```
 // CPP code to find the difference between highest
// and least frequencies
#include <bits/stdc++.h>
using namespace std;

int findDiff(int arr[], int n)
```

```
{
    // sort the array
    sort(arr, arr + n);

    int count = 0, max_count = 0, min_count = n;
    for (int i = 0; i < (n - 1); i++) {

        // checking consecutive elements
        if (arr[i] == arr[i + 1]) {
            count += 1;
            continue;
        }
        else {
            max_count = max(max_count, count);
            min_count = min(min_count, count);
            count = 0;
        }
    }

    return (max_count - min_count);
}

// Driver
int main()
{
    int arr[] = { 7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findDiff(arr, n) << "\n";
    return 0;
}
```

Java

```
 // JAVA Code for Difference between
// highest and least frequencies
// in an array
import java.util.*;

class GFG {

    static int findDiff(int arr[], int n)
    {
        // sort the array
        Arrays.sort(arr);

        int count = 0, max_count = 0,
            min_count = n;
```

```
        for (int i = 0; i < (n - 1); i++) {

            // checking consecutive elements
            if (arr[i] == arr[i + 1]) {
                count += 1;
                continue;
            }
            else {
                max_count = Math.max(max_count,
                                     count);

                min_count = Math.min(min_count,
                                     count);
                count = 0;
            }
        }

        return (max_count - min_count);
    }

    // Driver program to test above function
    public static void main(String[] args)
    {

        int arr[] = { 7, 8, 4, 5, 4, 1,
                      1, 7, 7, 2, 5 };
        int n = arr.length;

        System.out.println(findDiff(arr, n));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 code to find the difference
# between highest nd least frequencies

def findDiff(arr, n):

    # sort the array
    arr.sort()

    count = 0; max_count = 0; min_count = n
    for i in range(0, (n-1)):
```

```
        # checking consecutive elements
        if arr[i] == arr[i + 1]:
            count += 1
            continue
        else:
            max_count = max(max_count, count)
            min_count = min(min_count, count)
            count = 0
    return max_count - min_count

# Driver Code
arr = [ 7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5 ]
n = len(arr)
print (findDiff(arr, n))

# This code is contributed by Shreyanshi Arun.
```

C#

```
 // C# Code for Difference between
// highest and least frequencies
// in an array
using System;

class GFG {

    static int findDiff(int[] arr, int n)
    {

        // sort the array
        Array.Sort(arr);

        int count = 0, max_count = 0,
            min_count = n;

        for (int i = 0; i < (n - 1); i++) {

            // checking consecutive elements
            if (arr[i] == arr[i + 1]) {
                count += 1;
                continue;
            }
            else {
                max_count = Math.Max(max_count,
                                     count);

                min_count = Math.Min(min_count,
                                     count);
```

```
                count = 0;
            }
        }

        return (max_count - min_count);
    }

    // Driver program to test above function
    public static void Main()
    {

        int[] arr = { 7, 8, 4, 5, 4, 1,
                      1, 7, 7, 2, 5 };
        int n = arr.Length;

        Console.WriteLine(findDiff(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP code to find the
// difference between highest
// and least frequencies

// function that
// returns difference
function findDiff($arr, $n)
{

    // sort the array
    sort($arr);

    $count = 0; $max_count = 0;
    $min_count = $n;
    for ($i = 0; $i < ($n - 1); $i++)
    {

        // checking consecutive elements
        if ($arr[$i] == $arr[$i + 1])
        {
            $count += 1;
            continue;
        }
        else
```

```
        {
            $max_count = max($max_count, $count);
            $min_count = min($min_count, $count);
            $count = 0;
        }
    }

    return ($max_count - $min_count);
}

// Driver Code
$arr = array(7, 8, 4, 5, 4, 1,
             1, 7, 7, 2, 5);
$n = sizeof($arr);

echo(findDiff($arr, $n) . "\n");

// This code is contributed by Ajit.
?>
```

Output:

```
 2
```

An **efficient solution** is to use hashing. We count frequencies of all elements and finally traverse the hash table to find maximum and minimum.

Below is c++ implementation

```
 // CPP code to find the difference between highest
// and least frequencies
#include <bits/stdc++.h>
using namespace std;

int findDiff(int arr[], int n)
{
    // Put all elements in a hash map
    unordered_map<int, int> hm;
    for (int i = 0; i < n; i++)
        hm[arr[i]]++;

    // Find counts of maximum and minimum
    // frequent elements
    int max_count = 0, min_count = n;
    for (auto x : hm) {
        max_count = max(max_count, x.second);
        min_count = min(min_count, x.second);
```

```
    }

    return (max_count - min_count);
}

// Driver
int main()
{
    int arr[] = { 7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findDiff(arr, n) << "\n";
    return 0;
}
```

Output:

```
 2
```

Time Complexity : O(n)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/difference-between-highest-and-least-frequencies-in-an-array/

Chapter 66

Different substrings in a string that start and end with given strings

Different substrings in a string that start and end with given strings - GeeksforGeeks

Given a string s and two other strings begin and end, find the number of different substrings in the string which begin and end with the given begin and end strings.

Examples:

```
Input : s = "geeksforgeeks"
        begin = "geeks"
        end = "for"
Output : 1

Input : s = "vishakha"
        begin = "h"
        end = "a"
Output : 2
Two different sub-strings are "ha" and "hakha".
```

Approach : Find all occurrences of string begin and string end. Store the index of each string in two different arrays. After that traverse through whole string and add one symbol per iteration to already seen sub-strings and map new strings to some non-negative integers. As the ends and beginnings of strings and different string of equal length are mapped to different numbers (and equal strings are mapped equally), simply count the number of necessary sub-strings of certain length.

C++

```
 // Cpp program to find number of
// different sub stings
#include <bits/stdc++.h>
using namespace std;

// function to return number of different
// sub-strings
int numberOfDifferentSubstrings(string s, string a,
                                        string b)
{
    // initially our answer is zero.
    int ans = 0;

    // find the length of given strings
    int ls = s.size(), la = a.size(), lb = b.size();

    // currently make array and initially puy zero.
    int x[ls] = { 0 }, y[ls] = { 0 };

    // find occurence of "a" and "b" in string "s"
    for (int i = 0; i < ls; i++) {
        if (s.substr(i, la) == a)
            x[i] = 1;
        if (s.substr(i, lb) == b)
            y[i] = 1;
    }

    // We use a hash to make sure that same
    // substring is not counted twice.
    unordered_set<string> hash;

    // go through all the positions to find
    // occurrence of "a" first.
    string curr_substr = "";
    for (int i = 0; i < ls; i++) {

        // if we found occurrence of "a".
        if (x[i]) {

            // then go through all the positions
            // to find occurrence of "b".
            for (int j = i; j < ls; j++) {

                // if we do found "b" at index
                // j then add it to already
                // existed substring.
                if (!y[j])
                    curr_substr += s[j];
```

```
                // if we found occurrence of "b".
                if (y[j]) {

                    // now add string "b" to
                    // already existed substing.
                    curr_substr += s.substr(j, lb);

                    // If current substring is not
                    // included already.
                    if (hash.find(curr_substr) == hash.end())
                        ans++;

                    // put any non negative
                    // integer to make this
                    // string as already
                    // existed.
                    hash.insert(curr_substr);
                }
            }

            // make substring null.
            curr_substr = "";
        }
    }

    // return answer.
    return ans;
}

// Driver program for above function.
int main()
{
    string s = "codecppforfood";
    string begin = "c";
    string end = "d";
    cout << numberOfDifferentSubstrings(s, begin, end)
        << endl;
    return 0;
}
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/different-substrings-in-a-string-that-start-and-end-with-given-strings/

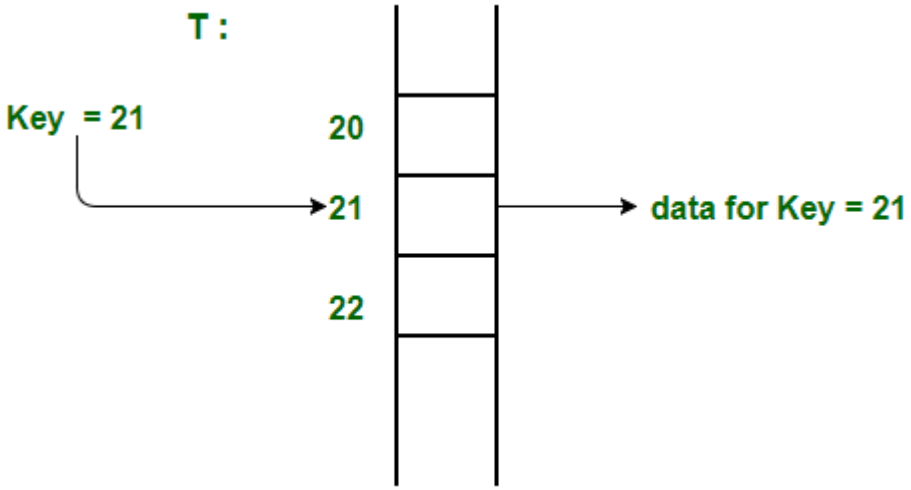
Chapter 67

Direct Address Table

Direct Address Table - GeeksforGeeks

Direct Address Table is a data structure that has the capability of mapping records to their corresponding keys using arrays. In direct address tables, records are placed using their key values directly as indexes. They facilitate fast searching, insertion and deletion operations.

We can understand the concept using the following example. We create an array of size equal to maximum value plus one (assuming 0 based index) and then use values as indexes. For example, in the following diagram key 21 is used directly as index.



Advantages:

1. **Searching in O(1) Time:** Direct address tables use arrays which are random access data structure, so, the key values (which are also the index of the array) can be easily used to search the records in O(1) time.

2. **Insertion in O(1) Time:** We can easily insert an element in an array in O(1) time. The same thing follows in a direct address table also.
3. **Deletion in O(1) Time:** Deletion of an element takes O(1) time in an array. Similarly, to delete an element in a direct address table we need O(1) time.

Limitations:

1. Prior knowledge of maximum key value
2. Practically useful only if the maximum value is very less.
3. It causes wastage of memory space if there is a significant difference between total records and maximum value.

Hashing can overcome these limitations of direct address tables.

How to handle collisions?
Collisions can be handled like Hashing. We can either use Chaining or open addressing to handle collisions. The only difference from hashing here is, we do not use a hash function to find the index. We rather directly use values as indexes.

Improved By : Srilekha_Paul

Source

https://www.geeksforgeeks.org/direct-address-table/

Chapter 68

Distinct Prime Factors of Array Product

Distinct Prime Factors of Array Product - GeeksforGeeks

Given an array of integers.Let us say P is the product of elements of the array. Find the number of of distinct prime factors of product P.

Examples:

> Input : 1 2 3 4 5
> Output : 3
> Explanation: Here P = 1 * 2 * 3 * 4 * 5 = 120. Distinct prime divisors of 120 are 2, 3 and 5. So, the output is 3.
>
> Input : 21 30 15 24 16
> Output : 4
> Explanation: Here P = 21 * 30 * 15 * 24 * 16 = 3628800. Distinct prime divisors of 3628800 are 2, 3, 5 and 7. So, the output is 4.

Naive Approach :
The simple solution for the problem would be to multiply every number in the array an then find the number of distinct prime factors of the product.
But this method can lead to integer overflow.

Better Approach :
To avoid the overflow instead of multiplying the numbers we can find the prime factors of each element separately and store the prime factors in a set or a map for unique factors.

C++

```
 // C++ program to count distinct prime
// factors of a number.
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to count the number of distinct prime
// factors of product of array
int Distinct_Prime_factors(vector<int> a)
{
    // use set to store distinct factors
    unordered_set<int> m;

    // iterate over every element of array
    for (int i = 0; i < a.size(); i++) {
        int sq = sqrt(a[i]);

        // from 2 to square root of number run
        // a loop and check the numbers which
        // are factors.
        for (int j = 2; j <= sq; j++) {
            if (a[i] % j == 0) {

                // if j is a factor store it in the set
                m.insert(j);

                // divide the number with j till it
                // is divisible so that only prime factors
                // are stored
                while (a[i] % j == 0) {
                    a[i] /= j;
                }
            }
        }

        // if the number is still greater than 1 then
        // it is a prime factor, insert in set
        if (a[i] > 1) {
            m.insert(a[i]);
        }
    }

    // the number of unique prime factors will
    // the size of the set
    return m.size();
}

// Driver Function
int main()
{
    vector<int> a = { 1, 2, 3, 4, 5 };
    cout << Distinct_Prime_factors(a) << '\n';
```

```
    return 0;
}
```

Java

```
// Java program to count distinct
// prime factors of a number.
import java.util.*;

class GFG
{

// Function to count the number
// of distinct prime factors of
// product of array
static int Distinct_Prime_factors(Vector a)
{
// use set to store distinct factors
HashSet m = new HashSet();

// iterate over every element of array
for (int i = 0; i < a.size(); i++) { int sq = (int)Math.sqrt(a.get(i)); // from 2 to square
root of number // run a loop and check the numbers // which are factors. for (int j = 2;
j <= sq; j++) { if (a.get(i) % j == 0) { // if j is a factor store // it in the set m.add(j);
// divide the number with j // till it is divisible so // that only prime factors // are stored
while (a.get(i) % j == 0) { a.set(i, a.get(i) / j); } } } // if the number is still greater //
than 1 then it is a prime factor, // insert in set if (a.get(i) > 1)
{
m.add(a.get(i));
}
}

// the number of unique prime
// factors will the size of the set
return m.size();
}

// Driver Code
public static void main(String args[])
{
Vector a = new Vector();
a.add(1);
a.add(2);
a.add(3);
a.add(4);
a.add(5);
System.out.println(Distinct_Prime_factors(a));
}
}

// This code is contributed by Arnab Kundu
```

Output :

```
3
```

Improved By : andrew1234

Source

https://www.geeksforgeeks.org/distinct-prime-factors-of-array-product/

Chapter 69

Distinct strings with odd and even changes allowed

Distinct strings with odd and even changes allowed - GeeksforGeeks

Given an array of lower case strings, the task is to find the number of strings that are distinct. Two strings are distinct if on applying the following operations on one string the second string cannot be formed.

- A character on odd index can be swapped with another character at odd index only.
- A character on even index can be swapped with another character on even index only.

Examples:

```
Input  : arr[] = {"abcd", "cbad", "bacd"}
Output : 2
The 2nd string can be converted to the 1st by swapping
the first and third characters. So there are 2 distinct
strings as the third string cannot be converted to the
first.

Input  : arr[] = {"abc", "cba"}
Output : 1
```

A **simple solution** is to run two loops. The outer loop picks a string and inner loop checks if there is a previously string which can be converted to current string by doing allowed transformations. This solution requires $O(n^2m)$ time where n is number of strings and m is maximum number of characters in any string.

An **efficient solution** generate an encoded string for every input string. The encoded has counts of even and odd positioned characters separated by a separator. Two strings

are considered distinct if their encoded strings are same else not. Once we have a way to encode string, the problem reduces to counting distinct encoded strings. This is a typical problem of hashing. We create hash set and one by one store encodings of strings. If an encoding already exists, we ignore string. Else we store encoding in hash and increment count of distinct strings.

C/C++

```
 // C++ program to count distinct strings with
// even odd swapping allowed.
#include<bits/stdc++.h>
using namespace std;
const int MAX_CHAR = 26;

// Returns encoding of string that can be used for hashing.
// The idea is to return same encoding for strings which
// can become same after swapping a even positioned character
// with other even characters OR swapping an odd character
// with other odd characters.
string encodeString(string str)
{
    // hashEven stores the count of even indexed character
    // for each string hashOdd stores the count of odd
    // indexed characters for each string
    int hashEven[MAX_CHAR] = {0};
    int hashOdd[MAX_CHAR] = {0};

    // creating hash for each string
    for (int i=0; i<str.length(); i++)
    {
        char c = str[i];
        if (i&1) // If index of current character is odd
           hashOdd[c-'a']++;
        else
           hashEven[c-'a']++;
    }

    // For every character from 'a' to 'z', we store its
    // count at even position followed by a separator,
    // followed by count at odd position.
    string encoding = "";
    for (int i=0; i<MAX_CHAR; i++)
    {
       encoding += to_string(hashEven[i]);
       encoding += to_string('-');
       encoding += to_string(hashOdd[i]);
       encoding += to_string('-');
    }
```

```
    return encoding;
}

// This function basically uses a hashing based set to
// store strings which are distinct according to accoding
// to criteria given in question.
int countDistinct(string input[], int n)
{
    int countDist = 0;  // Initialize result

    // Create an empty set and store all distinct
    // strings in it.
    unordered_set<string> s;
    for (int i=0; i<n; i++)
    {
       // If this encoding appears first time, increment
       // count of distinct encodings.
       if (s.find(encodeString(input[i])) == s.end())
       {
           s.insert(encodeString(input[i]));
           countDist++;
       }
    }

    return countDist;
}

// Driver code
int main()
{
    string input[] = {"abcd", "acbd", "adcb", "cdba",
                      "bcda", "badc"};
    int n = sizeof(input)/sizeof(input[0]);

    cout << countDistinct(input, n);
    return 0;
}
```

Java

```
 // Java program to count distinct strings with
// even odd swapping allowed.
import java.util.HashSet;
import java.util.Set;
class GFG {
static int MAX_CHAR = 26;

    static String encodeString(char[] str) {
```

```
        // hashEven stores the count of even indexed character
        // for each string hashOdd stores the count of odd
        // indexed characters for each string
        int hashEven[] = new int[MAX_CHAR];
        int hashOdd[] = new int[MAX_CHAR];

        // creating hash for each string
        for (int i = 0; i < str.length; i++) {
            char c = str[i];
            if ((i & 1) != 0) // If index of current character is odd
                hashOdd[c-'a']++;
            else
                hashEven[c-'a']++;

        }


        // For every character from 'a' to 'z', we store its
        // count at even position followed by a separator,
        // followed by count at odd position.
        String encoding = "";
        for (int i = 0; i < MAX_CHAR; i++) {
            encoding += (hashEven[i]);
            encoding += ('-');
            encoding += (hashOdd[i]);
            encoding += ('-');
        }
        return encoding;
    }

    // This function basically uses a hashing based set to
// store strings which are distinct according to accoding
// to criteria given in question.
    static int countDistinct(String input[], int n) {
        int countDist = 0; // Initialize result

        // Create an empty set and store all distinct
        // strings in it.
        Set<String> s = new HashSet<>();
        for (int i = 0; i < n; i++) {
            // If this encoding appears first time, increment
            // count of distinct encodings.
            if (!s.contains(encodeString(input[i].toCharArray()))) {
                s.add(encodeString(input[i].toCharArray()));
                countDist++;
            }
        }
```

```
        return countDist;
    }

    public static void main(String[] args) {
        String input[] = {"abcd", "acbd", "adcb", "cdba",
                "bcda", "badc"};
        int n = input.length;

        System.out.println(countDistinct(input, n));
    }
}
```

Output :

```
4
```

Improved By : krutikkhandhadiya

Source

https://www.geeksforgeeks.org/distinct-strings-odd-even-changes-allowed/

Chapter 70

Distributing items when a person cannot take more than two items of same type

Distributing items when a person cannot take more than two items of same type - GeeksforGeeks

Given N sweets which can be of many different types and k customers, one customer won't take the same type of sweet more than 2 pieces, the task is to find if it is possible to distribute all sweets then print "Yes" otherwise "No".

Given array arr[] represents array of sweets **arr[i] is type of sweet**.

Examples:

```
Input : arr[] = {1, 1, 2, 3, 1},
            k = 2;
Output : Yes
There are three pieces of sweet type 1,
one piece of type 2 and one piece of
type 3. Two customers can distribute
sweets under given constraints.

Input : arr[] = {2, 3, 3, 5, 3, 3},
            k = 2;
Output : No
```

Method 1:
1- Traverse array for each element.
2- Count occurrences of each element in array
3- Check Resulting occurrence of each element must be less than or equal to 2*k.

C++

```
 // C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to check occurrence of each element
bool checkCount(int arr[], int n, int k)
{
    int count;

    // Start traversing the elements
    for (int i = 0; i < n; i++) {

        // Count occurrences of current element
        count = 0;
        for (int j = 0; j < n; j++) {
            if (arr[j] == arr[i])
                count++;

            // If count of any element is greater
            // than 2*k then return false
            if (count > 2 * k)
                return false;
        }
    }

    return true;
}

// Drivers code
int main()
{
    int arr[] = { 1, 1, 2, 3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    checkCount(arr, n, k) ? cout << "Yes"
                          : cout << "No";

    return 0;
}
```

Java

```
 // java program for above implementation
import java.io.*;
```

```
public class GFG {

    // Function to check occurrence of
    // each element
    static boolean checkCount(int []arr,
                              int n, int k)
    {
        int count;

        // Start traversing the elements
        for (int i = 0; i < n; i++)
        {

            // Count occurrences of
            // current element
            count = 0;
            for (int j = 0; j < n; j++)
            {
                if (arr[j] == arr[i])
                    count++;

                // If count of any element
                // is greater than 2*k then
                // return false
                if (count > 2 * k)
                    return false;
            }
        }

        return true;
    }

    // Drivers code
    static public void main (String[] args)
    {
        int []arr = { 1, 1, 2, 3, 1 };
        int n = arr.length;
        int k = 2;

        if(checkCount(arr, n, k))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# program for above implementation
using System;

public class GFG {

    // Function to check occurrence
    // of each element
    static bool checkCount(int []arr,
                           int n, int k)
    {
        int count;

        // Start traversing the elements
        for (int i = 0; i < n; i++)
        {

            // Count occurrences of
            // current element
            count = 0;
            for (int j = 0; j < n; j++)
            {
                if (arr[j] == arr[i])
                    count++;

                // If count of any element
                // is greater than 2*k then
                // return false
                if (count > 2 * k)
                    return false;
            }
        }

        return true;
    }

    // Drivers code
    static public void Main ()
    {
        int []arr = { 1, 1, 2, 3, 1 };
        int n = arr.Length;
        int k = 2;

        if(checkCount(arr, n, k))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
```

```
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for above implementation

// Function to check occurrence
// of each element
function checkCount($arr, $n, $k)
{
    $count;

    // Start traversing the elements
    for($i = 0; $i < $n; $i++)
    {

        // Count occurrences of
        // current element
        $count = 0;
        for($j = 0; $j < $n; $j++)
        {
            if ($arr[$j] == $arr[$i])
                $count++;

            // If count of any element
            // is greater than 2*k then
            // return false
            if ($count > 2 * $k)
                return false;
        }
    }

    return true;
}

    // Driver Code
    $arr = array(1, 1, 2, 3, 1);
    $n =count($arr);
    $k = 2;
    if(checkCount($arr, $n, $k))
        echo "Yes";
    else
        echo "No";
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
Yes
```

Time Complexity: O(n^2)

Method 2:
1. Maintain a hash for 32 different type of sweets.
2. Traverse an array and check for every arr[i]

```
hash[arr[i]] <= 2*k.
```

```
 // C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to check hash array
// corresponding to the given array
bool checkCount(int arr[], int n, int k)
{
    unordered_map<int, int> hash;

    // Maintain a hash
    for (int i = 0; i < n; i++)
        hash[arr[i]]++;

    // Check for each value in hash
    for (auto x : hash)
        if (x.second > 2 * k)
            return false;

    return true;
}

// Drivers code
int main()
{
    int arr[] = { 1, 1, 2, 3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    checkCount(arr, n, k) ? cout << "Yes"
                          : cout << "No";
    return 0;
}
```

Output:

```
Yes
```

Time Complexity : O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/distributing-items-person-cannot-take-two-items-type/

Chapter 71

Double Hashing

Double Hashing - GeeksforGeeks

Double hashing is a collision resolving technique in **Open Addressed** Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

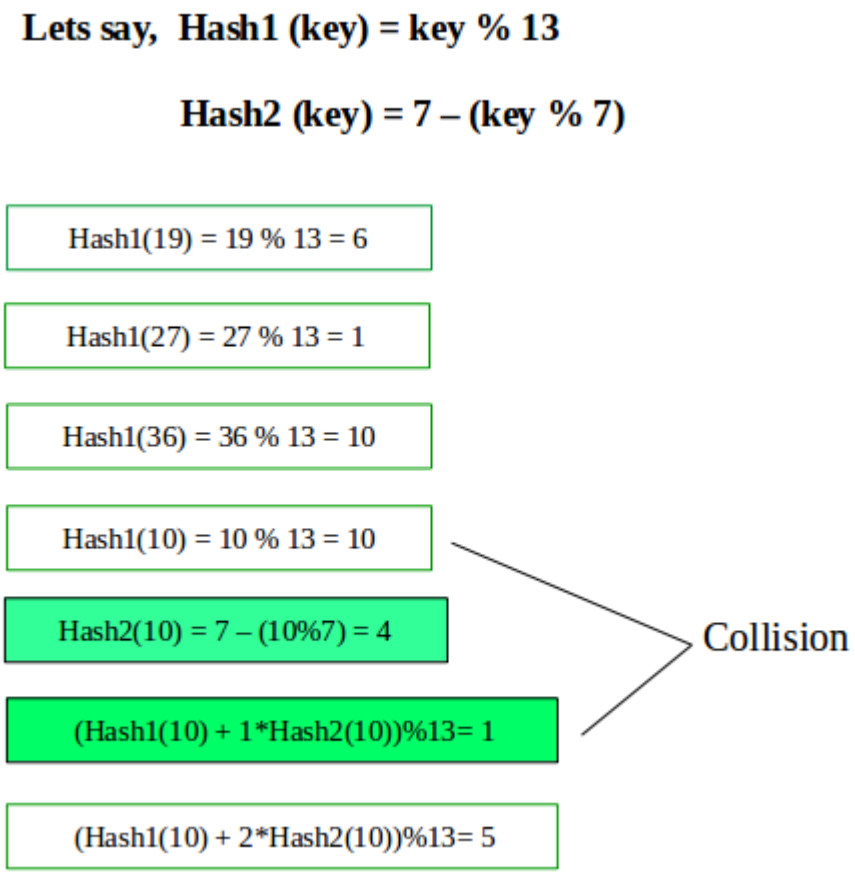
> Double hashing can be done using :
> **(hash1(key) + i * hash2(key)) % TABLE_SIZE**
> Here hash1() and hash2() are hash functions and TABLE_SIZE
> is size of hash table.
> (We repeat by increasing i when collision occurs)

First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is : **hash2(key) = PRIME – (key % PRIME)** where PRIME is a prime smaller than the TABLE_SIZE.

A good second Hash function is:

- It must never evaluate to zero
- Must make sure that all cells can be probed




```
 // CPP program to implement double hashing
#include <bits/stdc++.h>
using namespace std;

// Hash table size
#define TABLE_SIZE 13

// Used in second hash function.
#define PRIME 7

class DoubleHash
{
    // Pointer to an array containing buckets
    int *hashTable;
    int curr_size;

public:
```

```
// function to check if hash table is full
bool isFull()
{

    // if hash size reaches maximum size
    return (curr_size == TABLE_SIZE);
}

// function to calculate first hash
int hash1(int key)
{
    return (key % TABLE_SIZE);
}

// function to calculate second hash
int hash2(int key)
{
    return (PRIME - (key % PRIME));
}

DoubleHash()
{
    hashTable = new int[TABLE_SIZE];
    curr_size = 0;
    for (int i=0; i<TABLE_SIZE; i++)
        hashTable[i] = -1;
}

// function to insert key into hash table
void insertHash(int key)
{
    // if hash table is full
    if (isFull())
        return;

    // get index from first hash
    int index = hash1(key);

    // if collision occurs
    if (hashTable[index] != -1)
    {
        // get index2 from second hash
        int index2 = hash2(key);
        int i = 1;
        while (1)
        {
            // get newIndex
```

```
                int newIndex = (index + i * index2) %
                                        TABLE_SIZE;

                // if no collision occurs, store
                // the key
                if (hashTable[newIndex] == -1)
                {
                    hashTable[newIndex] = key;
                    break;
                }
                i++;
            }
        }

        // if no collision occurs
        else
            hashTable[index] = key;
        curr_size++;
    }

    // function to display the hash table
    void displayHash()
    {
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            if (hashTable[i] != -1)
                cout << i << " --> "
                     << hashTable[i] << endl;
            else
                cout << i << endl;
        }
    }
};

// Driver's code
int main()
{
    int a[] = {19, 27, 36, 10, 64};
    int n = sizeof(a)/sizeof(a[0]);

    // inserting keys into hash table
    DoubleHash h;
    for (int i = 0; i < n; i++)
        h.insertHash(a[i]);

    // display the hash Table
    h.displayHash();
    return 0;
```

```
}
```

Output:

```
0
1 --> 27
2
3
4
5 --> 10
6 --> 19
7
8
9
10 --> 36
11
12 --> 64
```

Source

https://www.geeksforgeeks.org/double-hashing/

Chapter 72

Efficiently find first repeated character in a string without using any additional data structure in one traversal

Efficiently find first repeated character in a string without using any additional data structure in one traversal - GeeksforGeeks

Implement a space efficient algorithm to check First repeated character in a string without using any additional data structure in one traversal. Use additional data structures like count array, hash, etc is not allowed.

Examples :

```
Input :  abcfdeacf
Output : char = a, index= 6
```

The idea is to use an integer variable and uses bits in its binary representation to store whether a character is present or not. Typically an integer has at-least 32 bits and we need to store presence/absence of only 26 characters.

C++

```
 // Efficiently check First repeated character
// in C++ program
#include<bits/stdc++.h>
using namespace std;

// Returns -1 if all characters of str are
```

```
// unique.
// Assumptions : (1) str contains only characters
//                   from 'a' to 'z'
//               (2) integers are stored using 32
//                   bits
int FirstRepeated(string str)
{
    // An integer to store presence/absence
    // of 26 characters using its 32 bits.
    int checker = 0;

    for (int i = 0; i < str.length(); ++i)
    {
        int val = (str[i]-'a');

        // If bit corresponding to current
        // character is already set
        if ((checker & (1 << val)) > 0)
            return i;

        // set bit in checker
        checker |= (1 << val);
    }

    return -1;
}

// Driver code
int main()
{
    string s = "abcfdeacf";
    int i=FirstRepeated(s);
    if (i!=-1)
        cout <<"Char = "<< s[i] << "   and Index = "<<i;
    else
        cout << "No repeated Char";
    return 0;
}
```

Java

```
 // Efficiently check First repeated character
// in Java program
public class First_Repeated_char {

    static int FirstRepeated(String str)
    {
        // An integer to store presence/absence
```

```
        // of 26 characters using its 32 bits.
        int checker = 0;

        for (int i = 0; i < str.length(); ++i)
        {
            int val = (str.charAt(i)-'a');

            // If bit corresponding to current
            // character is already set
            if ((checker & (1 << val)) > 0)
                return i;

            // set bit in checker
            checker |= (1 << val);
        }

        return -1;
    }

    // Driver code
    public static void main(String args[])
    {
        String s = "abcfdeacf";
        int i=FirstRepeated(s);
        if (i!=-1)
           System.out.println("Char = "+ s.charAt(i) + "   and Index = "+i);
        else
            System.out.println( "No repeated Char");
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Efficiently check First repeated character
# in Python

# Returns -1 if all characters of str are
# unique.
# Assumptions : (1) str contains only characters
#                 from 'a' to 'z'
##             (2) integers are stored using 32
##                 bits
def FirstRepeated(string):

    # An integer to store presence/absence
    # of 26 characters using its 32 bits.
    checker = 0
```

```
    pos = 0
    for i in string:
        val = ord(i) - ord('a');

        # If bit corresponding to current
        # character is already set
        if ((checker & (1 << val)) > 0):
            return pos

        # set bit in checker
        checker |= (1 << val)
        pos += 1

    return -1

# Driver code
string = "abcfdeacf"
i = FirstRepeated(string)
if i != -1:
    print "Char = ", string[i], " and Index = ", i;
else:
    print "No repeated Char"

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to Efficiently
// check First repeated character
using System;

public class First_Repeated_char {

    static int FirstRepeated(string str)
    {
        // An integer to store presence/absence
        // of 26 characters using its 32 bits.
        int checker = 0;

        for (int i = 0; i < str.Length; ++i)
        {
            int val = (str[i]-'a');

            // If bit corresponding to current
            // character is already set
            if ((checker & (1 << val)) > 0)
                return i;
```

```
            // set bit in checker
            checker |= (1 << val);
        }

        return -1;
    }

    // Driver code
    public static void Main()
    {
        string s = "abcfdeacf";
        int i=FirstRepeated(s);
        if (i!=-1)
           Console.WriteLine("Char = " + s[i] +
                          " and Index = " + i);
        else
            Console.WriteLine( "No repeated Char");
    }
}

// This code is contributed by vt_m.
```

Output:

```
Char = a   and Index = 6
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/efficiently-find-first-repeated-character-string-without-using-additional-data-structu

Chapter 73

Elements of first array that have more frequencies

Elements of first array that have more frequencies - GeeksforGeeks

Given two arrays (which may or may not be sorted). There arrays are such that they might have some common elements in them. We need to find elements whose counts of occurrences are more in first array than second.

Examples:

```
Input : ar1[] = {1, 2, 2, 2, 3, 3, 4, 5}
        ar2[] = {2, 2, 3, 3, 3, 4}
Output : 1 2 5
1 occurs one times in first and zero times in second
2 occurs three times in first and two times in second
.........................

Input : ar1[] = {1, 3, 4, 2, 3}
        ar2[] = {3, 4, 5}
Output : 3
```

The idea is to use hashing. We traverse first array and insert all elements and their frequencies in a hash table. Now we traverse through the second array and reduce frequencies in hash table for the common elements. Now we traverse through first array again and print those elements whose frequencies are still more than 0. To avoid repeated printing of same elements, we set frequency as 0.

```
 // C++ program to print all those elements of
// first array that have more frequencies than
// second array.
#include <bits/stdc++.h>
```

```
using namespace std;

// Compares two intervals according to staring times.
void moreFreq(int ar1[], int ar2[], int m, int n)
{
    // Traverse first array and store frequencies
    // of all elements
    unordered_map<int, int> mp;
    for (int i = 0; i < m; i++)
        mp[ar1[i]]++;

    // Traverse second array and reduce frequencies
    // of common elements.
    for (int i = 0; i < n; i++)
        if (mp.find(ar2[i]) != mp.end())
            mp[ar2[i]]--;

    // Now traverse first array again and print
    // all those elements whose frequencies are
    // more than 0. To avoid repeated printing,
    // we set frequency as 0 after printing.
    for (int i = 0; i < m; i++) {
        if (mp[ar1[i]] > 0) {
            cout << ar1[i] << " ";
            mp[ar1[i]] = 0;
        }
    }
}

// Driver code
int main()
{
    int ar1[] = { 1, 2, 2, 2, 3, 3, 4, 5 };
    int ar2[] = { 2, 2, 3, 3, 3, 4 };
    int m = sizeof(ar1) / sizeof(ar1[0]);
    int n = sizeof(ar2) / sizeof(ar2[0]);
    moreFreq(ar1, ar2, m, n);
    return 0;
}
```

Output:

```
1 2 5
```

Time Complexity : O(m + n) under the assumption that unordered_map find() and insert() work in O(1) time.

Source

https://www.geeksforgeeks.org/elements-of-first-array-that-have-more-frequencies/

Chapter 74

Elements that occurred only once in the array

Elements that occurred only once in the array - GeeksforGeeks

Given an array *arr* that has numbers appearing twice or once. The task is to identify numbers that occurred only once in the array.

Note: Duplicates appear side by side every time. Might be few numbers can occur one time and just assume this is a right rotating array (just say an array can rotate k times towards right). Order of the elements in the output doesn't matter.

Examples:

```
Input: arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 }
Output: 9 4

Input: arr[] = {-9, -8, 4, 4, 5, 5, -1}
Output: -9 -8 -1
```

Method-1: *Using Sorting.*

- Sort the array.
- Check for each element at index i (except the first and last element), if

  ```
  arr[i] != arr[i-1] && arr [i] != arr[i+1]
  ```

- For the first element, check if ***arr[0] != arr[1].***
- For the last element, check if ***arr[n-1] != arr[n-2].***

Below is the implementation of above approach:

C++

```
 // C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the elements that
// appeared only once in the array
void occurredOnce(int arr[], int n)
{
    // Sort the array
    sort(arr, arr + n);

    // Check for first element
    if (arr[0] != arr[1])
        cout << arr[0] << " ";

    // Check for all the elements if it is different
    // its adjacent elements
    for (int i = 1; i < n - 1; i++)
        if (arr[i] != arr[i + 1] && arr[i] != arr[i - 1])
            cout << arr[i] << " ";

    // Check for the last element
    if (arr[n - 2] != arr[n - 1])
        cout << arr[n - 1] << " ";
}

// Driver code
int main()
{

    int arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    occurredOnce(arr, n);

    return 0;
}
```

Java

```
 // Java implementation
// of above approach
import java.util.*;
```

```
class GFG
{

// Function to find the elements that
// appeared only once in the array
static void occurredOnce(int arr[], int n)
{
    // Sort the array
    Arrays.sort(arr);

    // Check for first element
    if (arr[0] != arr[1])
        System.out.println(arr[0] + " ");

    // Check for all the elements
    // if it is different
    // its adjacent elements
    for (int i = 1; i < n - 1; i++)
        if (arr[i] != arr[i + 1] &&
            arr[i] != arr[i - 1])
            System.out.print(arr[i] + " ");

    // Check for the last element
    if (arr[n - 2] != arr[n - 1])
        System.out.print(arr[n - 1] + " ");
}

// Driver code
public static void main(String args[])
{
    int arr[] = {7, 7, 8, 8, 9,
                 1, 1, 4, 2, 2};
    int n = arr.length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by Arnab Kundu
```

Python 3

```
 # Python 3 implementation
# of above approach

# Function to find the elements
# that appeared only once in
```

```
# the array
def occurredOnce(arr, n):

    # Sort the array
    arr.sort()

    # Check for first element
    if arr[0] != arr[1]:
        print(arr[0], end = " ")

    # Check for all the elements
    # if it is different its
    # adjacent elements
    for i in range(1, n - 1):
        if (arr[i] != arr[i + 1] and
            arr[i] != arr[i - 1]):
            print( arr[i], end = " ")

    # Check for the last element
    if arr[n - 2] != arr[n - 1]:
        print(arr[n - 1], end = " ")

# Driver code
if __name__ == "__main__":

    arr = [ 7, 7, 8, 8, 9,
            1, 1, 4, 2, 2 ]
    n = len(arr)
    occurredOnce(arr, n)

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# implementation
// of above approach
using System;

class GFG
{

// Function to find the elements that
// appeared only once in the array
static void occurredOnce(int[] arr, int n)
{
    // Sort the array
    Array.Sort(arr);
```

```
    // Check for first element
    if (arr[0] != arr[1])
        Console.Write(arr[0] + " ");

    // Check for all the elements
    // if it is different
    // its adjacent elements
    for (int i = 1; i < n - 1; i++)
        if (arr[i] != arr[i + 1] &&
            arr[i] != arr[i - 1])
            Console.Write(arr[i] + " ");

    // Check for the last element
    if (arr[n - 2] != arr[n - 1])
        Console.Write(arr[n - 1] + " ");
}

// Driver code
public static void Main()
{
    int[] arr = {7, 7, 8, 8, 9,
                 1, 1, 4, 2, 2};
    int n = arr.Length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP implementation
// of above approach

// Function to find the elements
// that appeared only once in
// the array
function occurredOnce(&$arr, $n)
{
    // Sort the array
    sort($arr);

    // Check for first element
    if ($arr[0] != $arr[1])
```

```
        echo $arr[0]." ";

    // Check for all the elements
    // if it is different its
    // adjacent elements
    for ($i = 1; $i < $n - 1; $i++)
        if ($arr[$i] != $arr[$i + 1] &&
            $arr[$i] != $arr[$i - 1])
            echo $arr[$i]." ";

    // Check for the last element
    if ($arr[$n - 2] != $arr[$n - 1])
        echo $arr[$n - 1]." ";
}

// Driver code
$arr = array(7, 7, 8, 8, 9,
             1, 1, 4, 2, 2);
$n = sizeof($arr);
occurredOnce($arr, $n);

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
4 9
```

Time Complexity: O(Nlogn)
Space Complexity: O(1)

Method-2: *(Using Hashing)*: In C++, unordered_map can be used for hashing.

- Traverse the array.
- Store each element with its occurrence in the unordered_map.
- Traverse the unordered_map and print all the elements with occurrence 1.

Below is the implementation of above approach:

C++

```
 // C++ implementation to find elements
// that appeared only once
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to find the elements that
// appeared only once in the array
void occurredOnce(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Store all the lements int the map with
    // theri occurrence
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Traverse the map and print all the
    // elements with occurrence 1
    for (auto it = mp.begin(); it != mp.end(); it++)
        if (it->second == 1)
            cout << it->first << " ";
}

// Driver code
int main()
{

    int arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    occurredOnce(arr, n);

    return 0;
}
```

Output:

```
4 9
```

Time Complexity: O(N)
Space Complexity: O(N)

Method-3: *Using given assumptions.*
It is given that an array can be rotated any times and duplicates will appear side by side every time. So after rotating the first and last element will appear side by side.

- Check if the first and last element is equal. If yes then start traversing the elements between them.
- Check if the current element is equal to the element at immediate previous index. If yes, check the same for next element.

- If not, print the current element.

C++

```
 // C++ implementation to find elements
// that appeared only once
#include <bits/stdc++.h>
using namespace std;

// Function to find the elements that
// appeared only once in the array
void occurredOnce(int arr[], int n)
{
    int i = 1, len = n;

    // Check if the first and last element is equal
    // If yes, remove those elements
    if (arr[0] == arr[len - 1]) {
        i = 2;
        len--;
    }

    // Start traversing the remaining elements
    for (; i < n; i++)

        // Check if current element is equal to
        // the element at immediate previous index
        // If yes, check the same for next element
        if (arr[i] == arr[i - 1])
            i++;

        // Else print the current element
        else
            cout << arr[i - 1] << " ";

    // Check for the last element
    if (arr[n - 1] != arr[0] && arr[n - 1] != arr[n - 2])
        cout << arr[n - 1];
}

// Driver code
int main()
{

    int arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    occurredOnce(arr, n);
```

```
    return 0;
}
```

Java

```
 // Java implementation to find
// elements that appeared only once
class GFG
{
// Function to find the elements that
// appeared only once in the array
static void occurredOnce(int arr[], int n)
{
    int i = 1, len = n;

    // Check if the first and last
    // element is equal. If yes,
    // remove those elements
    if (arr[0] == arr[len - 1])
    {
        i = 2;
        len--;
    }

    // Start traversing the
    // remaining elements
    for (; i < n; i++)

        // Check if current element is
        // equal to the element at
        // immediate previous index
        // If yes, check the same
        // for next element
        if (arr[i] == arr[i - 1])
            i++;

        // Else print the current element
        else
            System.out.print(arr[i - 1] + " ");

    // Check for the last element
    if (arr[n - 1] != arr[0] &&
        arr[n - 1] != arr[n - 2])
        System.out.print(arr[n - 1]);
}

// Driver code
```

```
public static void main(String args[])
{
    int arr[] = {7, 7, 8, 8, 9,
                 1, 1, 4, 2, 2};
    int n = arr.length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by Arnab Kundu
```

Python 3

```
 # Python 3 implementation to find
# elements that appeared only once

# Function to find the elements that
# appeared only once in the array
def occurredOnce(arr, n):
    i = 1
    len = n

    # Check if the first and
    # last element is equal
    # If yes, remove those elements
    if arr[0] == arr[len - 1]:
        i = 2
        len -= 1

    # Start traversing the
    # remaining elements
    while i < n:

        # Check if current element is
        # equal to the element at
        # immediate previous index
        # If yes, check the same for
        # next element
        if arr[i] == arr[i - 1]:
            i += 1

        # Else print the current element
        else:
            print(arr[i - 1], end = " ")

        i += 1
```

```
    # Check for the last element
    if (arr[n - 1] != arr[0] and
        arr[n - 1] != arr[n - 2]):
        print(arr[n - 1])

# Driver code
if __name__ == "__main__":
    arr = [ 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 ]
    n = len(arr)

    occurredOnce(arr, n)

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# implementation to find
// elements that appeared only once
using System;

class GFG
{
// Function to find the elements that
// appeared only once in the array
static void occurredOnce(int[] arr, int n)
{
    int i = 1, len = n;

    // Check if the first and last
    // element is equal. If yes,
    // remove those elements
    if (arr[0] == arr[len - 1])
    {
        i = 2;
        len--;
    }

    // Start traversing the
    // remaining elements
    for (; i < n; i++)

        // Check if current element is
        // equal to the element at
        // immediate previous index
        // If yes, check the same
        // for next element
```

```
        if (arr[i] == arr[i - 1])
            i++;

        // Else print the current element
        else
            Console.Write(arr[i - 1] + " ");

    // Check for the last element
    if (arr[n - 1] != arr[0] &&
        arr[n - 1] != arr[n - 2])
        Console.Write(arr[n - 1]);
}

// Driver code
public static void Main()
{
    int[] arr = {7, 7, 8, 8, 9,
                 1, 1, 4, 2, 2};
    int n = arr.Length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP implementation to find
// elements that appeared only once

// Function to find the elements that
// appeared only once in the array
function occurredOnce(&$arr, $n)
{
    $i = 1;
    $len = $n;

    // Check if the first and last
    // element is equal. If yes,
    // remove those elements
    if ($arr[0] == $arr[$len - 1])
    {
        $i = 2;
        $len--;
    }
```

```
    // Start traversing the
    // remaining elements
    for (; $i < $n; $i++)

        // Check if current element is
        // equal to the element at
        // immediate previous index
        // If yes, check the same for
        // next element
        if ($arr[$i] == $arr[$i - 1])
            $i++;

        // Else print the current element
        else
            echo $arr[$i - 1] . " ";

    // Check for the last element
    if ($arr[$n - 1] != $arr[0] &&
        $arr[$n - 1] != $arr[$n - 2])
        echo $arr[$n - 1];
}

// Driver code
$arr = array(7, 7, 8, 8, 9,
             1, 1, 4, 2, 2);
$n = sizeof($arr);

occurredOnce($arr, $n);

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
9 4
```

Time Complexity: O(N)
Space Complexity: O(1)

Improved By : andrew1234, ChitraNayal

Source

https://www.geeksforgeeks.org/elements-that-occurred-only-once-in-the-array/

Chapter 75

Elements to be added so that all elements of a range are present in array

Elements to be added so that all elements of a range are present in array - GeeksforGeeks

Given an array of size N. Let A and B be the minimum and maximum in the array respectively. Task is to find how many number should be added to the given array such that all the element in the range [A, B] occur at-least once in the array.

Examples:

```
Input : arr[] = {4, 5, 3, 8, 6}
Output : 1
Only 7 to be added in the list.

Input : arr[] = {2, 1, 3}
Output : 0
```

Method 1 (Sorting)
1- Sort the array.
2- Compare arr[i] == arr[i+1]-1 or not. If not, update count = arr[i+1]-arr[i]-1.
3- Return count.

C++

```
 // C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to count numbers to be added
int countNum(int arr[], int n)
{
    int count = 0;

    // Sort the array
    sort(arr, arr + n);

    // Check if elements are consecutive
    //  or not. If not, update count
    for (int i = 0; i < n - 1; i++)
        if (arr[i] != arr[i+1] &&
            arr[i] != arr[i + 1] - 1)
            count += arr[i + 1] - arr[i] - 1;

    return count;
}

// Drivers code
int main()
{
    int arr[] = { 3, 5, 8, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countNum(arr, n) << endl;
    return 0;
}
```

Java

```
 // java program for above implementation
import java.io.*;
import java.util.*;

public class GFG {

    // Function to count numbers to be added
    static int countNum(int []arr, int n)
    {
        int count = 0;

        // Sort the array
        Arrays.sort(arr);

        // Check if elements are consecutive
        // or not. If not, update count
        for (int i = 0; i < n - 1; i++)
            if (arr[i] != arr[i+1] &&
                arr[i] != arr[i + 1] - 1)
```

```
                count += arr[i + 1] - arr[i] - 1;

        return count;
    }

    // Drivers code
    static public void main (String[] args)
    {

        int []arr = { 3, 5, 8, 6 };
        int n = arr.length;

        System.out.println(countNum(arr, n));
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # python program for above implementation

# Function to count numbers to be added
def countNum(arr, n):

    count = 0

    # Sort the array
    arr.sort()

    # Check if elements are consecutive
    # or not. If not, update count
    for i in range(0, n-1):
        if (arr[i] != arr[i+1] and
            arr[i] != arr[i + 1] - 1):
            count += arr[i + 1] - arr[i] - 1;

    return count

# Drivers code
arr = [ 3, 5, 8, 6 ]
n = len(arr)
print(countNum(arr, n))

# This code is contributed by Sam007
```

C#

```
 // C# program for above implementation
using System;

public class GFG {

    // Function to count numbers to be added
    static int countNum(int []arr, int n)
    {
        int count = 0;

        // Sort the array
        Array.Sort(arr);

        // Check if elements are consecutive
        // or not. If not, update count
        for (int i = 0; i < n - 1; i++)
            if (arr[i] != arr[i+1] &&
                arr[i] != arr[i + 1] - 1)
                count += arr[i + 1] - arr[i] - 1;

        return count;
    }

    // Drivers code
    static public void Main ()
    {

        int []arr = { 3, 5, 8, 6 };
        int n = arr.Length;

        Console.WriteLine(countNum(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for
// above implementation

// Function to count
// numbers to be added
function countNum($arr, $n)
{

    $count = 0;
```

```
    // Sort the array
    sort($arr);

    // Check if elements are
    // consecutive or not.
    // If not, update count
    for ($i = 0; $i < $n - 1; $i++)
        if ($arr[$i] != $arr[$i + 1] &&
            $arr[$i] != $arr[$i + 1] - 1)
            $count += $arr[$i + 1] -
                      $arr[$i] - 1;

    return $count;
}

// Driver code
$arr = array(3, 5, 8, 6);
$n = count($arr);
echo countNum($arr, $n) ;

// This code is contributed
// by anuj_67.
?>
```

Output:

```
2
```

Time Complexity: O(n log n)

Method 2 (Use Hashing)
1- Maintain a hash of array elements.
2- Store minimm and maximum element.
3- Traverse from minimum to maximum element in hash
And count if element is not in hash.
4- Return count.

C++

```
 // C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to count numbers to be added
int countNum(int arr[], int n)
```

```
{
    unordered_set<int> s;
    int count = 0, maxm = INT_MIN, minm = INT_MAX;

    // Make a hash of elements
    // and store minimum and maximum element
    for (int i = 0; i < n; i++) {
        s.insert(arr[i]);
        if (arr[i] < minm)
            minm = arr[i];
        if (arr[i] > maxm)
            maxm = arr[i];
    }

    // Traverse all elements from minimum
    // to maximum and count if it is not
    // in the hash
    for (int i = minm; i <= maxm; i++)
        if (s.find(arr[i]) == s.end())
            count++;
    return count;
}

// Drivers code
int main()
{
    int arr[] = { 3, 5, 8, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countNum(arr, n) << endl;
    return 0;
}
```

Output:

```
2
```

Time Complexity- O(max – min + 1)

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/elements-to-be-added-so-that-all-elements-of-a-range-are-present-in-array/

Chapter 76

Equally divide into two sets such that one set has maximum distinct elements

Equally divide into two sets such that one set has maximum distinct elements - GeeksforGeeks

There are two processes P1 and P2, and N resources where N is an even number. There is an array of N size and arr[i] represents the type of ith resource.There may be more then one instance of a resource.You are to divide these resources equally between P1 and P2 such that maximum number of distinct number of resources are allocated to P2. Print maximum number of distinct resources allocated to P2.

Examples:

> Input : arr[] = [1, 1, 2, 2, 3, 3]
> Output: 3
> Explanation:
> There are three different kinds of resources (1, 2 and 3), and two for each kind.
> Optimal distribution: Process P1 has resources [1, 2, 3] and the process P2 has gifts [1, 2, 3], too. Process p2 has 3 distinct resources.
>
> arr[] = [1, 1, 2, 1, 3, 4]
> Output: 3
> Explanation:
> There are three different kinds of resources (1, 2, 3, 4), 3 instances of 1 and single single instances of resource 2, 3, 4. Optimal distribution: Process P1 has resources [1, 1, 1] and the process P2 has gifts [2, 3, 4].
> Process p2 has 3 distinct resources.

Approach 1 (Using sorting):
1. Sort the Array of resources.

2. Find out the elements which are unique by comparing the adjacent elements of the sorted array.suppose count holds the distinct number of resources in array.
3. Return the minimum of count and N/2.

Time complexity- O(N log N)

C++

```
 // C++ program to equally divide n elements
// into two sets such that second set has
// maximum distinct elements.
#include <algorithm>
#include <iostream>
using namespace std;

int distribution(int arr[], int n)
{
    sort(arr, arr + n);
    int count = 1;
    for (int i = 1; i < n; i++)
        if (arr[i] > arr[i - 1])
            count++;

    return min(count, n / 2);
}

// Driver code
int main()
{
    int arr[] = { 1, 1, 2, 1, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << distribution(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to equally divide n elements
// into two sets such that second set has
// maximum distinct elements.
import java.util.*;
class Geeks {

static int distribution(int arr[], int n)
{
    Arrays.sort(arr);
    int count = 1;
    for (int i = 1; i < n; i++)
```

```
        if (arr[i] > arr[i - 1])
            count++;

    return Math.min(count, n / 2);
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 1, 2, 1, 3, 4 };
    int n = arr.length;
    System.out.println(distribution(arr, n));
}
}

// This code is contributed by ankita_saini
```

C#

```
 // C# program to equally divide
// n elements into two sets such
// that second set has maximum
// distinct elements.
using System;

class GFG
{
static int distribution(int []arr, int n)
{
    Array.Sort(arr);
    int count = 1;
    for (int i = 1; i < n; i++)
        if (arr[i] > arr[i - 1])
            count++;

    return Math.Min(count, n / 2);
}

// Driver code
public static void Main(String []args)
{
    int []arr= { 1, 1, 2, 1, 3, 4 };
    int n = arr.Length;
    Console.WriteLine(distribution(arr, n));
}
}

// This code is contributed
```

```
// by ankita_saini
```

Ouput:

```
3
```

Approach 2(using hash set)
Another way to find out distinct element is set, insert all the element in the set. By the property of a set, it will contain only unique elements. At the end, we can count the number of elements in the set, given by, say count. The value to be returned will again be given by min(count, n/2).

C++

```
 // C++ program to equally divide n elements
// into two sets such that second set has
// maximum distinct elements.
#include <bits/stdc++.h>
using namespace std;

int distribution(int arr[], int n)
{
    unordered_set<int, greater<int> > resources;

    // Insert all the resources in the set
    // There will be unique resources in the set
    for (int i = 0; i < n; i++)
        resources.insert(arr[i]);

    // return minimum of distinct resources
    // and n/2
    return min(resources.size(), n / 2);
}

// Driver code
int main()
{
    int arr[] = { 1, 1, 2, 1, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << distribution(arr, n) << endl;
    return 0;
}
```

Improved By : ankita_saini

Source

https://www.geeksforgeeks.org/equally-divide-into-two-sets-such-that-one-set-has-maximum-distinct-elements/

Chapter 77

Find All Duplicate Subtrees

Find All Duplicate Subtrees - GeeksforGeeks

Given a binary tree, find all duplicate subtrees. For each duplicate subtrees, we only need to return the root node of any one of them. Two trees are duplicate if they have the same structure with same node values.

Examples:

```
Input :
       1
      / \
     2   3
    /   / \
   4   2   4
      /
     4

Output :
   2
  /    and    4
 4
Explanation: Above Trees are two duplicate subtrees.
Therefore, you need to return above trees root in the
form of a list.
```

The idea is to use hashing. We store inorder traversals of subtrees in a hash. Since simple inorder traversal cannot uniquely identify a tree, we use symbols like '(' and ')' to represent NULL nodes.
We pass a Unordered Map in C++ as an argument to the helper function which recursively calculates inorder string and increases its count in map. If any string gets repeated, then it will imply duplication of the subtree rooted at that node so push that node in Final result and return the vector of these nodes.

C++

```
 // C++ program to find averages of all levels
// in a binary tree.
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to
left child and a pointer to right child */
struct Node {
    int data;
    struct Node* left, *right;
};

string inorder(Node* node, unordered_map<string, int>& m)
{
    if (!node)
        return "";

    string str = "(";
    str += inorder(node->left, m);
    str += to_string(node->data);
    str += inorder(node->right, m);
    str += ")";

    // Subtree already present (Note that we use
    // unordered_map instead of unordered_set
    // because we want to print multiple duplicates
    // only once, consider example of 4 in above
    // subtree, it should be printed only once.
    if (m[str] == 1)
        cout << node->data << " ";

    m[str]++;

    return str;
}

// Wrapper over inorder()
void printAllDups(Node* root)
{
    unordered_map<string, int> m;
    inorder(root, m);
}

/* Helper function that allocates a
new node with the given data and
NULL left and right pointers. */
```

```
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver code
int main()
{
    Node* root = NULL;
    root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->right->left = newNode(2);
    root->right->left->left = newNode(4);
    root->right->right = newNode(4);
    printAllDups(root);
    return 0;
}
```

Java

```
 // A java program to find all duplicate subtrees
// in a binary tree.
import java.util.HashMap;
public class Duplicate_subtress {

    /* A binary tree node has data, pointer to
    left child and a pointer to right child */
    static HashMap<String, Integer> m;
    static class Node {
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            left = null;
            right = null;
        }
    }
    static String inorder(Node node)
    {
        if (node == null)
            return "";
```

```
        String str = "(";
        str += inorder(node.left);
        str += Integer.toString(node.data);
        str += inorder(node.right);
        str += ")";

        // Subtree already present (Note that we use
        // HashMap instead of HashSet
        // because we want to print multiple duplicates
        // only once, consider example of 4 in above
        // subtree, it should be printed only once.
        if (m.get(str) != null && m.get(str)==1 )
            System.out.print( node.data + " ");

        if (m.containsKey(str))
            m.put(str, m.get(str) + 1);
        else
            m.put(str, 1);


        return str;
    }

    // Wrapper over inorder()
    static void printAllDups(Node root)
    {
        m = new HashMap<>();
        inorder(root);
    }
    // Driver code
    public static void main(String args[])
    {
        Node root = null;
        root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.right.left = new Node(2);
        root.right.left.left = new Node(4);
        root.right.right = new Node(4);
        printAllDups(root);
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
4 2
```

Source

https://www.geeksforgeeks.org/find-duplicate-subtrees/

Chapter 78

Find Itinerary from a given list of tickets

Find Itinerary from a given list of tickets - GeeksforGeeks

Given a list of tickets, find itinerary in order using the given list.

Example:

```
Input:
"Chennai" -> "Banglore"
"Bombay" -> "Delhi"
"Goa"    -> "Chennai"
"Delhi"  -> "Goa"

Output:
Bombay->Delhi, Delhi->Goa, Goa->Chennai, Chennai->Banglore,
```

It may be assumed that the input list of tickets is not cyclic and there is one ticket from every city except final destination.

One Solution is to build a graph and do Topological Sorting of the graph. Time complexity of this solution is O(n).

We can also use hashingto avoid building a graph. The idea is to first find the starting point. A starting point would never be on 'to' side of a ticket. Once we find the starting point, we can simply traverse the given map to print itinerary in order. Following are steps.

```
1) Create a HashMap of given pair of tickets.  Let the created
   HashMap be 'dataset'. Every entry of 'dataset' is of the form
   "from->to" like "Chennai" -> "Banglore"

2) Find the starting point of itinerary.
     a) Create a reverse HashMap.  Let the reverse be 'reverseMap'
```

```
        Entries of 'reverseMap' are of the form "to->form".
        Following is 'reverseMap' for above example.
        "Banglore"-> "Chennai"
        "Delhi"   -> "Bombay"
        "Chennai" -> "Goa"
        "Goa"     ->  "Delhi"

     b) Traverse 'dataset'.  For every key of dataset, check if it
        is there in 'reverseMap'.  If a key is not present, then we
        found the starting point. In the above example, "Bombay" is
        starting point.

3) Start from above found starting point and traverse the 'dataset'
   to print itinerary.
```

All of the above steps require O(n) time so overall time complexity is O(n).

Below is Java implementation of above idea.

Java

```
 // Java program to print itinerary in order
import java.util.HashMap;
import java.util.Map;

public class printItinerary
{
    // Driver function
    public static void main(String[] args)
    {
        Map<String, String> dataSet = new HashMap<String, String>();
        dataSet.put("Chennai", "Banglore");
        dataSet.put("Bombay", "Delhi");
        dataSet.put("Goa", "Chennai");
        dataSet.put("Delhi", "Goa");

        printResult(dataSet);
    }

    // This function populates 'result' for given input 'dataset'
    private static void printResult(Map<String, String> dataSet)
    {
        // To store reverse of given map
        Map<String, String> reverseMap = new HashMap<String, String>();

        // To fill reverse map, iterate through the given map
        for (Map.Entry<String,String> entry: dataSet.entrySet())
            reverseMap.put(entry.getValue(), entry.getKey());
```

```
        // Find the starting point of itinerary
        String start = null;
        for (Map.Entry<String,String> entry: dataSet.entrySet())
        {
              if (!reverseMap.containsKey(entry.getKey()))
              {
                   start = entry.getKey();
                   break;
              }
        }

        // If we could not find a starting point, then something wrong
        // with input
        if (start == null)
        {
           System.out.println("Invalid Input");
           return;
        }

        // Once we have starting point, we simple need to go next, next
        // of next using given hash map
        String to = dataSet.get(start);
        while (to != null)
        {
            System.out.print(start +  "->" + to + ", ");
            start = to;
            to = dataSet.get(to);
        }
    }
}
```

C++

```
 #include <iostream>
#include <map>
#include <string>
using namespace std;

void printItinerary(map<string, string> dataSet)
{
    // To store reverse of given map
    map<string, string> reversemap;
    map<string, string>::iterator it;

    // To fill reverse map, iterate through the given map
    for (it = dataSet.begin(); it!=dataSet.end(); it++)
        reversemap[it->second] = it->first;
```

```
    // Find the starting point of itinerary
    string start;

    for (it = dataSet.begin(); it!=dataSet.end(); it++)
    {
        if (reversemap.find(it->first) == reversemap.end())
        {
            start = it->first;
            break;
        }
    }

    // If we could not find a starting point, then something wrong with input
     if (start.empty())
     {
        cout << "Invalid Input" << endl;
        return;
     }

    // Once we have starting point, we simple need to go next,
    //next of next using given hash map
    it = dataSet.find(start);
    while (it != dataSet.end())
    {
        cout << it->first << "->" << it->second << endl;
        it = dataSet.find(it->second);
    }

}

int main()
{
    map<string, string> dataSet;
    dataSet["Chennai"] = "Banglore";
    dataSet["Bombay"] = "Delhi";
    dataSet["Goa"] = "Chennai";
    dataSet["Delhi"] = "Goa";

    printItinerary(dataSet);

    return 0;
}
// C++ implementation is contributed by Aditya Goel
```

Output:

```
Bombay->Delhi, Delhi->Goa, Goa->Chennai, Chennai->Banglore,
```

This article is compiled by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/find-itinerary-from-a-given-list-of-tickets/

Chapter 79

Find Recurring Sequence in a Fraction

Find Recurring Sequence in a Fraction - GeeksforGeeks

Given a fraction, find recurring sequence of digits if exists, otherwise print “No recurring sequence”.

Examples:

```
Input  : Numerator = 8, Denominator = 3
Output : Recurring sequence is 6
Explanation : 8/3 = 2.66666666.......

Input : Numerator = 50, Denominator = 22
Output : Recurring sequence is 27
Explanation : 50/22 = 2.272727272.....

Input : Numerator = 11, Denominator = 2
Output : No recurring sequence
Explanation : 11/2 = 5.5
```

When does the fractional part repeat ?

Let us simulate the process of converting fraction to decimal. Let us look at the part where we have already figured out the integer part which is floor(numerator/denominator). Now we are left with (remainder = numerator%denominator) / denominator.
If you remember the process of converting to decimal, at each step we do the following :

1. Multiply the remainder by 10.
2. Append remainder / denominator to result.
3. Remainder = remainder % denominator.

At any moment, if remainder becomes 0, we are done.

However, when there is a recurring sequence, remainder never becomes 0. For example if you look at 1/3, the remainder never becomes 0.

Below is one important observation :
If we start with remainder 'rem' and if the remainder repeats at any point of time, the digits between the two occurrence of 'rem' keep repeating.

So the idea is to store seen remainders in a map. Whenever a remainder repeats, we return the sequence before the next occurrence. Below is C++ implementation of above idea.

```
 // C++ program to find repeating sequence in a fraction
#include <bits/stdc++.h>
using namespace std;

// This function returns repeating sequence of
// a fraction.  If repeating sequence doesn't
// exits, then returns empty string
string fractionToDecimal(int numr, int denr)
{
    string res; // Initialize result

    // Create a map to store already seen remainders
    // remainder is used as key and its position in
    // result is stored as value. Note that we need
    // position for cases like 1/6.  In this case,
    // the recurring sequence doesn't start from first
    // remainder.
    map <int, int> mp;
    mp.clear();

    // Find first remainder
    int rem = numr%denr;

    // Keep finding remainder until either remainder
    // becomes 0 or repeats
    while ( (rem!=0) && (mp.find(rem) == mp.end()) )
    {
        // Store this remainder
        mp[rem] = res.length();

        // Multiply remainder with 10
        rem = rem*10;

        // Append rem / denr to result
        int res_part = rem / denr;
        res += to_string(res_part);
```

```
        // Update remainder
        rem = rem % denr;
    }

    return (rem == 0)? "" : res.substr(mp[rem]);
}

// Driver code
int main()
{
    int numr = 50, denr = 22;
    string res = fractionToDecimal(numr, denr);
    if (res == "")
        cout << "No recurring sequence";
    else
        cout << "Recurring sequence is " << res;
    return 0;
}
```

Output :

```
Recurring sequence is 27
```

This article is contributed by **Dhruv Mahajan**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/find-recurring-sequence-fraction/

Chapter 80

Find Sum of all unique sub-array sum for a given array.

Find Sum of all unique sub-array sum for a given array. - GeeksforGeeks

Given an array of n-positive elements. Sub-array sum is defined as the sum of all elements of a particular sub-array, the task is to find the sum of all unique sub-array sum.
Note: Unique Sub-array sum means no other sub-array will have the same sum value.

Examples:

> Input : arr[] = {3, 4, 5}
> Output : 36
> Explanation: All possible unique sub-array with their sum are as:
> (3), (4), (5), (3+4), (4+5), (3+4+5). Here all are unique so required sum = 36
>
> Input : arr[] = {2, 4, 2}
> Output : 12
> Explanation: All possible unique sub-array with their sum are as:
> (2), (4), (2), (2+4), (4+2), (2+4+2). Here only (4) and (2+4+2) are unique.

Method 1 (Sorting Based)
1- Calculate cumulative sum of array.
2- Store all sub-array sum in vector.
3- Sort the vector.
4- Mark all duplicate sub-array sum to zero
5- Calculate and return totalSum.

```
 // CPP for finding sum of all unique subarray sum
#include <bits/stdc++.h>
using namespace std;

// function for finding grandSum
```

```
long long int findSubarraySum(int arr[], int n)
{
    int i, j;

    // calculate cumulative sum of array
    // cArray[0] will store sum of zero elements
    long long int cArray[n + 1] = { 0 };
    for (i = 0; i < n; i++)
        cArray[i + 1] = cArray[i] + arr[i];

    vector<long long int> subArrSum;

    // store all subarray sum in vector
    for (i = 1; i <= n; i++)
        for (j = i; j <= n; j++)
            subArrSum.push_back(cArray[j] - cArray[i - 1]);

    // sort the vector
    sort(subArrSum.begin(), subArrSum.end());

    // mark all duplicate sub-array sum to zero
    long long totalSum = 0;
    for (i = 0; i < subArrSum.size() - 1; i++) {
        if (subArrSum[i] == subArrSum[i + 1]) {
            j = i + 1;
            while (subArrSum[j] == subArrSum[i] && j < subArrSum.size()) {
                subArrSum[j] = 0;
                j++;
            }
            subArrSum[i] = 0;
        }
    }

    // calculate total sum
    for (i = 0; i < subArrSum.size(); i++)
        totalSum += subArrSum[i];

    // return totalSum
    return totalSum;
}

// Drivers code
int main()
{
    int arr[] = { 3, 2, 3, 1, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findSubarraySum(arr, n);
    return 0;
```

```
}
```

Output:

```
41
```

Method 2 (Hashing Based) The idea is make an empty hash table. We generate all subarrays. For every subarray, we compute its sum and increment count of sum in hash table. Finally we add all those sums whose count is 1.

```
 // CPP for finding sum of all unique subarray sum
#include <bits/stdc++.h>
using namespace std;

// function for finding grandSum
long long int findSubarraySum(int arr[], int n)
{
    int res = 0;

    // Go through all subarrays, compute sums
    // and count occurrences of sums.
    unordered_map<int, int> m;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += arr[j];
            m[sum]++;
        }
    }

    // Print all those sums that appear
    // once.
    for (auto x : m)
        if (x.second == 1)
            res += x.first;

    return res;
}

// Driver code
int main()
{
    int arr[] = { 3, 2, 3, 1, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findSubarraySum(arr, n);
    return 0;
}
```

Output:

```
41
```

Source

https://www.geeksforgeeks.org/find-sum-unique-sub-array-sum-given-array/

Chapter 81

Find a pair of elements swapping which makes sum of two arrays same

Find a pair of elements swapping which makes sum of two arrays same - GeeksforGeeks

Given two arrays of integers, find a pair of values (one value from each array) that you can swap to give the two arrays the same sum.

Examples:

```
Input : A[] = {4, 1, 2, 1, 1, 2}
        B[] = (3, 6, 3, 3)
Output : {1, 3}
Sum of elements in A[] = 11
Sum of elements in B[] = 15
To get same sum from both arrays, we
can swap following values:
1 from A[] and 3 from B[]

Input: A[] = {5, 7, 4, 6}
       B[] = {1, 2, 3, 8}
Output: 6 2
```

Method 1 (Naive Implementation)
Iterate through the arrays and check all pairs of values. Compare new sums or look for a pair with that difference.

C/C++

```
 // CPP code naive solution to find a pair swapping
// which makes sum of arrays sum.
#include <iostream>
using namespace std;

// Function to calculate sum of elements of array
int getSum(int X[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += X[i];
    return sum;
}

void findSwapValues(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // Look for val1 and val2, such that
    // sumA - val1 + val2 = sumB - val2 + val1
    int newsum1, newsum2, val1, val2;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            newsum1 = sum1 - A[i] + B[j];
            newsum2 = sum2 - B[j] + A[i];
            if (newsum1 == newsum2) {
                val1 = A[i];
                val2 = B[j];
            }
        }
    }

    cout << val1 << "  " << val2;
}

// Driver code
int main()
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = sizeof(A) / sizeof(A[0]);
    int B[] = { 3, 6, 3, 3 };
    int m = sizeof(B) / sizeof(B[0]);

    // Call to function
    findSwapValues(A, n, B, m);
    return 0;
```

```
}
```

Java

```
 // Java program to find a pair swapping
// which makes sum of arrays sum
import java.io.*;

class GFG
{
    // Function to calculate sum of elements of array
    static int getSum(int X[], int n)
    {
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += X[i];
        return sum;
    }

    // Function to prints elements to be swapped
    static void findSwapValues(int A[], int n, int B[], int m)
    {
        // Calculation of sums from both arrays
        int sum1 = getSum(A, n);
        int sum2 = getSum(B, m);

        // Look for val1 and val2, such that
        // sumA - val1 + val2 = sumB - val2 + val1
        int newsum1, newsum2, val1 = 0, val2 = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                newsum1 = sum1 - A[i] + B[j];
                newsum2 = sum2 - B[j] + A[i];
                if (newsum1 == newsum2)
                {
                    val1 = A[i];
                    val2 = B[j];
                }
            }
        }

        System.out.println(val1+" "+val2);
    }

    // driver program
    public static void main (String[] args)
```

```
    {
        int A[] = { 4, 1, 2, 1, 1, 2 };
        int n = A.length;
        int B[] = { 3, 6, 3, 3 };
        int m = B.length;

        // Call to function
        findSwapValues(A, n, B, m);
    }
}

// Contributed by Pramod Kumar
```

Python

```
 # Python code naive solution to find a pair swapping
# which makes sum of lists sum.

# Function to calculate sum of elements of list
def getSum(X):
    sum=0
    for i in X:
        sum+=i
    return sum

# Function to prints elements to be swapped
def findSwapValues(A,B):
    # Calculation if sums from both lists
    sum1=getSum(A)
    sum2=getSum(B)

    # Boolean variable used to reduce further iterations
    # after the pair is found
    k=False

    # Lool for val1 and val2, such that
    # sumA - val1 + val2 = sumB -val2 + val1
    val1,val2=0,0
    for i in A:
        for j in B:
            newsum1=sum1-i+j
            newsum2=sum2-j+i

            if newsum1 ==newsum2:
                val1=i
                val2=j
                # Set to True when pair is found
                k=True
```

```
                break
        # If k is True, it means pair is found.
        # So, no further iterations.
        if k==True:
            break
    print val1,val2
    return


# Driver code
A=[4,1,2,1,1,2]
B=[3,6,3,3]

# Call to function
findSwapValues(A,B)

# code contributed by sachin bisht
```

Output :

```
1 3
```

Time Complexity :- O(n*m)

Method 2 -> Other Naive implementation

```
We are looking for two values, a and b, such that:
sumA - a + b = sumB - b + a
    2a - 2b  = sumA - sumB
      a - b  = (sumA - sumB) / 2
```

Therefore, we're looking for two values that have a specific target difference: (sumA – sumB) / 2.

C/C++

```
 // CPP code for naive implementation
#include <iostream>
using namespace std;

// Function to calculate sum of elements of array
int getSum(int X[], int n)
{
```

```
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += X[i];
    return sum;
}

// Function to calculate : a - b = (sumA - sumB) / 2
int getTarget(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // because that the target must be an integer
    if ((sum1 - sum2) % 2 != 0)
        return 0;
    return ((sum1 - sum2) / 2);
}

void findSwapValues(int A[], int n, int B[], int m)
{
    int target = getTarget(A, n, B, m);
    if (target == 0)
        return;

    // Look for val1 and val2, such that
    // val1 - val2 = (sumA - sumB) / 2
    int val1, val2;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (A[i] - B[j] == target) {
                val1 = A[i];
                val2 = B[j];
            }
        }
    }

    cout << val1 << "  " << val2;
}

// Driver code
int main()
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = sizeof(A) / sizeof(A[0]);
    int B[] = { 3, 6, 3, 3 };
    int m = sizeof(B) / sizeof(B[0]);
```

```
    // Call to function
    findSwapValues(A, n, B, m);
    return 0;
}
```

Java

```
 // Java program to find a pair swapping
// which makes sum of arrays sum
import java.io.*;

class GFG
{
    // Function to calculate sum of elements of array
    static int getSum(int X[], int n)
    {
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += X[i];
        return sum;
    }

    // Function to calculate : a - b = (sumA - sumB) / 2
    static int getTarget(int A[], int n, int B[], int m)
    {
        // Calculation of sums from both arrays
        int sum1 = getSum(A, n);
        int sum2 = getSum(B, m);

        // because that the target must be an integer
        if ((sum1 - sum2) % 2 != 0)
            return 0;
        return ((sum1 - sum2) / 2);
    }

    // Function to prints elements to be swapped
    static void findSwapValues(int A[], int n, int B[], int m)
    {
        int target = getTarget(A, n, B, m);
        if (target == 0)
            return;

        // Look for val1 and val2, such that
        // val1 - val2 = (sumA - sumB) / 2
        int val1 = 0, val2 = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
```

```
            {
                if (A[i] - B[j] == target)
                {
                    val1 = A[i];
                    val2 = B[j];
                }
            }
        }
        System.out.println(val1+" "+val2);
    }

    // driver program
    public static void main (String[] args)
    {
        int A[] = { 4, 1, 2, 1, 1, 2 };
        int n = A.length;
        int B[] = { 3, 6, 3, 3 };
        int m = B.length;

        // Call to function
        findSwapValues(A, n, B, m);
    }
}

// Contributed by Pramod Kumar
```

Python

```
 # Python Code for naive implementation

# Function to calculate sum of elements of list
def getSum(X):
    sum=0
    for i in X:
        sum+=i
    return sum

# Function to calculate : a-b = (sumA - sumB) / 2
def getTarget(A,B):

    #Calculations of sums from both lists
    sum1=getSum(A)
    sum2=getSum(B)

    # Because the target must be an integer
    if( (sum1-sum2)%2!=0):
        return 0
    return (sum1-sum2)/2
```

```
def findSwapValues(A,B):
    target=getTarget(A,B)
    if target==0:
        return

    # Boolean variable used to reduce further iterations
    # after the pair is found
    flag=False

    # Look for val1 and val2, such that
    # val1 - val2 = (sumA -sumB) /2
    val1,val2=0,0
    for i in A:
        for j in B:

            if i-j == target:
                val1=i
                val2=j
                # Set to True when pair is found
                flag=True
                break
        if flag==True:
            break
    print val1,val2
    return


# Driver code
A=[4,1,2,1,1,2]
B=[3,6,3,3]

# Call to function
findSwapValues(A,B)

# code contributed by sachin bisht
```

Output:

```
1 3
```

Time Complexity :- O(n*m)

Method 3 -> Optimized Solution :-

```
1) Sort the arrays.
2) Traverse both array simultaneously and do
   following for every pair.
    a) If difference is too small then,
       make it bigger by moving 'a' to a
       bigger value.
    b) If it is too big then, make it smaller
       by moving b to a bigger value.
    c) If it's just right, return this pair.
```

C/C++

```
 // CPP code for optimized implementation
#include <bits/stdc++.h>
using namespace std;

// Returns sum of elements in X[]
int getSum(int X[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += X[i];
    return sum;
}

// Finds value of
// a - b = (sumA - sumB) / 2
int getTarget(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // because that the target must be an integer
    if ((sum1 - sum2) % 2 != 0)
        return 0;
    return ((sum1 - sum2) / 2);
}

// Prints elements to be swapped
void findSwapValues(int A[], int n, int B[], int m)
{
    // Call for sorting the arrays
    sort(A, A + n);
    sort(B, B + m);

    // Note that target can be negative
    int target = getTarget(A, n, B, m);
```

```
    // target 0 means, answer is not possible
    if (target == 0)
        return;

    int i = 0, j = 0;
    while (i < n && j < m) {
        int diff = A[i] - B[j];
        if (diff == target) {
            cout << A[i] << " " << B[j];
            return;
        }

        // Look for a greater value in A[]
        else if (diff < target)
            i++;

        // Look for a greater value in B[]
        else
            j++;
    }
}

// Driver code
int main()
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = sizeof(A) / sizeof(A[0]);

    int B[] = { 1, 6, 3, 3 };
    int m = sizeof(B) / sizeof(B[0]);

    findSwapValues(A, n, B, m);
    return 0;
}
```

Java

```
 // Java code for optimized implementation
import java.io.*;
import java.util.*;

class GFG
{
    // Function to calculate sum of elements of array
    static int getSum(int X[], int n)
    {
        int sum = 0;
```

```
    for (int i = 0; i < n; i++)
        sum += X[i];
    return sum;
}

// Function to calculate : a - b = (sumA - sumB) / 2
static int getTarget(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // because that the target must be an integer
    if ((sum1 - sum2) % 2 != 0)
        return 0;
    return ((sum1 - sum2) / 2);
}

// Function to prints elements to be swapped
static void findSwapValues(int A[], int n, int B[], int m)
{
    // Call for sorting the arrays
    Arrays.sort(A);
    Arrays.sort(B);

    // Note that target can be negative
    int target = getTarget(A, n, B, m);

    // target 0 means, answer is not possible
    if (target == 0)
        return;

    int i = 0, j = 0;
    while (i < n && j < m)
    {
        int diff = A[i] - B[j];
        if (diff == target)
        {
            System.out.println(A[i]+" "+B[i]);
            return;
        }

        // Look for a greater value in A[]
        else if (diff < target)
            i++;

        // Look for a greater value in B[]
        else
```

```
                j++;
        }
    }

    // driver program
    public static void main (String[] args)
    {
        int A[] = { 4, 1, 2, 1, 1, 2 };
        int n = A.length;
        int B[] = { 3, 6, 3, 3 };
        int m = B.length;

        // Call to function
        findSwapValues(A, n, B, m);
    }
}

// Contributed by Pramod Kumar
```

Python

```
 # Python code for optimized implementation

#Returns sum of elements in list
def getSum(X):
    sum=0
    for i in X:
        sum+=i
    return sum

# Finds value of
# a - b = (sumA - sumB) / 2
def getTarget(A,B):
    # Calculations of sumd from both lists
    sum1=getSum(A)
    sum2=getSum(B)

    # Because that target must be an integer
    if( (sum1-sum2)%2!=0):
        return 0
    return (sum1-sum2)/2

# Prints elements to be swapped
def findSwapValues(A,B):
    # Call for sorting the lists
    A.sort()
    B.sort()
```

```
    #Note that target can be negative
    target=getTarget(A,B)

    # target 0 means, answer is not possible
    if(target==0):
        return
    i,j=0,0
    while(i<len(A) and j<len(B)):
        diff=A[i]-B[j]
        if diff == target:
            print A[i],B[j]
            return
        # Look for a greater value in list A
        elif diff <target:
            i+=1
        # Look for a greater value in list B
        else:
            j+=1

A=[4,1,2,1,1,2]
B=[3,6,3,3]

findSwapValues(A,B)

#code contibuted by sachin bisht
```

Output:

```
2 3
```

Time Complexity :-
If arrays are sorted : **O(n + m)**
If arrays aren't sorted : **O(nlog(n) + mlog(m))**

Method 4 (Hashing)
We can solve this problem in **O(m+n) time** and O(m) auxiliary space. Below are algorithmic steps.

```
// assume array1 is small i.e. (m < n)
// where m is array1.length and n is array2.length
1. Find sum1(sum of small array elements) ans sum2
  (sum of larger array elements). // time O(m+n)
2. Make a hashset for small array(here array1).
3. Calculate diff as (sum1-sum2)/2.
```

```
4. Run a loop for array2
     for (int i equal to 0 to n-1)
       if (hashset contains (array2[i]+diff))
           print array2[i]+diff and array2[i]
           set flag  and break;
5. If flag is unset then there is no such kind of
pair.
```

Thanks to nicky khan for suggesting method 4.

Source

https://www.geeksforgeeks.org/find-a-pair-swapping-which-makes-sum-of-two-arrays-same/

Chapter 82

Find a pair with given sum in BST

Find a pair with given sum in BST - GeeksforGeeks

Given a BST and a sum, find if there is a pair with given sum.

```
Input : sum = 28
        Root of below tree

Output : Pair is found (16, 12)
```

We have discussed different approaches to find a pair with given sum in below post.Find a pair with given sum in a Balanced BST

In this post, hashing based solution is discussed. We traverse binary search tree by inorder way and insert node's value into a set. Also check for any node, difference between given sum and node's value in set, if it is found then pair exists otherwise it doesn't exist.

```
 // CPP program to find a pair with
// given sum using hashing
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    struct Node *left, *right;
};

Node* NewNode(int data)
{
    Node* temp = (Node*)malloc(sizeof(Node));
```

```
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

Node* insert(Node* root, int key)
{
    if (root == NULL)
        return NewNode(key);
    if (key < root->data)
        root->left = insert(root->left, key);
    else
        root->right = insert(root->right, key);
    return root;
}

bool findpairUtil(Node* root, int sum,  unordered_set<int> &set)
{
    if (root == NULL)
        return false;

    if (findpairUtil(root->left, sum, set))
        return true;

    if (set.find(sum - root->data) != set.end()) {
        cout << "Pair is found ("
             << sum - root->data << ", "
             << root->data << ")" << endl;
        return true;
    }
    else
        set.insert(root->data);

    return findpairUtil(root->right, sum, set);
}

void findPair(Node* root, int sum)
{
    unordered_set<int> set;
    if (!findpairUtil(root, sum, set))
        cout << "Pairs do not exit" << endl;
}

// Driver code
int main()
{
    Node* root = NULL;
```

```
    root = insert(root, 15);
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 8);
    root = insert(root, 12);
    root = insert(root, 16);
    root = insert(root, 25);
    root = insert(root, 10);

    int sum = 33;
    findPair(root, sum);

    return 0;
}
```

Output:

```
Pair is found (8, 25)
```

Time Complexity is O(n).

Source

https://www.geeksforgeeks.org/find-pair-given-sum-bst/

Chapter 83

Find all triplets with zero sum

Find all triplets with zero sum - GeeksforGeeks

Given an array of distinct elements. The task is to find triplets in array whose sum is zero.

Examples :

```
Input : arr[] = {0, -1, 2, -3, 1}
Output : 0 -1 1
         2 -3 1

Input : arr[] = {1, -2, 1, 0, 5}
Output : 1 -2  1
```

Method 1 (Simple : $O(n^3)$)
The naive approach is that run three loops and check one by one that sum of three elements is zero or not if sum of three elements is zero then print elements other wise print not found.

C++

```
 // A simple C++ program to find three elements
// whose sum is equal to zero
#include<bits/stdc++.h>
using namespace std;

// Prints all triplets in arr[] with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = true;
    for (int i=0; i<n-2; i++)
    {
```

```
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (arr[i]+arr[j]+arr[k] == 0)
                {
                    cout << arr[i] << " "
                         << arr[j] << " "
                         << arr[k] <<endl;
                    found = true;
                }
            }
        }
    }

    // If no triplet with 0 sum found in array
    if (found == false)
        cout << " not exist "<<endl;

}

// Driver code
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}
```

Java

```
 // A simple Java program to find three elements
// whose sum is equal to zero
class num{
// Prints all triplets in arr[] with 0 sum
static void findTriplets(int[] arr, int n)
{
    boolean found = true;
    for (int i=0; i<n-2; i++)
    {
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (arr[i]+arr[j]+arr[k] == 0)
                {
                    System.out.print(arr[i]);
```

```
                    System.out.print(" ");
                    System.out.print(arr[j]);
                    System.out.print(" ");
                    System.out.print(arr[k]);
                    System.out.print("\n");
                    found = true;
                }
            }
        }
    }

    // If no triplet with 0 sum found in array
    if (found == false)
        System.out.println(" not exist ");

}

// Driver code
public static void main(String[] args)
{
    int arr[] = {0, -1, 2, -3, 1};
    int n =arr.length;
    findTriplets(arr, n);

}
}
//This code is contributed by
//Smitha Dinesh Semwal
```

Python3

```
 # A simple Python 3 program
# to find three elements whose
# sum is equal to zero

# Prints all triplets in
# arr[] with 0 sum
def findTriplets(arr, n):

    found = True
    for i in range(0, n-2):

        for j in range(i+1, n-1):

            for k in range(j+1, n):

                if (arr[i] + arr[j] + arr[k] == 0):
                    print(arr[i], arr[j], arr[k])
```

```
                    found = True


    # If no triplet with 0 sum
    # found in array
    if (found == False):
        print(" not exist ")

# Driver code
arr = [0, -1, 2, -3, 1]
n = len(arr)
findTriplets(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // A simple C# program to find three elements
// whose sum is equal to zero
using System;

class GFG {

    // Prints all triplets in arr[] with 0 sum
    static void findTriplets(int []arr, int n)
    {
        bool found = true;
        for (int i = 0; i < n-2; i++)
        {
            for (int j = i+1; j < n-1; j++)
            {
                for (int k = j+1; k < n; k++)
                {
                    if (arr[i] + arr[j] + arr[k]
                                            == 0)
                    {
                        Console.Write(arr[i]);
                        Console.Write(" ");
                        Console.Write(arr[j]);
                        Console.Write(" ");
                        Console.Write(arr[k]);
                        Console.Write("\n");
                        found = true;
                    }
                }
            }
        }
```

```
        // If no triplet with 0 sum found in
        // array
        if (found == false)
            Console.Write(" not exist ");
    }

    // Driver code
    public static void Main()
    {
        int []arr = {0, -1, 2, -3, 1};
        int n = arr.Length;
        findTriplets(arr, n);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A simple PHP program to
// find three elements whose
// sum is equal to zero

// Prints all triplets
// in arr[] with 0 sum
function findTriplets($arr, $n)
{
    $found = true;
    for ($i = 0; $i < $n - 2; $i++)
    {
        for ($j = $i + 1; $j < $n - 1; $j++)
        {
            for ($k = $j + 1; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] +
                              $arr[$k] == 0)
                {
                    echo $arr[$i] , " ",
                         $arr[$j] , " ",
                         $arr[$k] ,"\n";
                    $found = true;
                }
            }
        }
    }

    // If no triplet with 0
```

```
    // sum found in array
    if ($found == false)
        echo " not exist ", "\n";

}

// Driver Code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n);

// This code is contributed by m_kit
?>


0 -1 1
2 -3 1
```

Time Complexity : $O(n^3)$
Auxiliary Space : $O(1)$

Method 2 (Hashing : $O(n^2)$)
We iterate through every element. For every element arr[i], we find a pair with sum "-arr[i]". This problem reduces to pairs sum and can be solved in O(n) time using hashing.

```
Run a loop from i=0 to n-2
  Create an empty hash table
  Run inner loop from j=i+1 to n-1
      If -(arr[i] + arr[j]) is present in hash table
         print arr[i], arr[j] and -(arr[i]+arr[j])
      Else
         Insert arr[j] in hash table.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is zero
#include<bits/stdc++.h>
using namespace std;

// function to print triplets with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = false;

    for (int i=0; i<n-1; i++)
```

```
    {
        // Find all pairs with sum equals to
        // "-arr[i]"
        unordered_set<int> s;
        for (int j=i+1; j<n; j++)
        {
            int x = -(arr[i] + arr[j]);
            if (s.find(x) != s.end())
            {
                printf("%d %d %d\n", x, arr[i], arr[j]);
                found = true;
            }
            else
                s.insert(arr[j]);
        }
    }

    if (found == false)
        cout << " No Triplet Found" << endl;
}

// Driver code
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}


-1 0 1
-3 2 1
```

Time Complexity : $O(n^2)$
Auxiliary Space : $O(n)$

Method 3 (Sorting : $O(n^2)$)

The above method requires extra space. We can solve in O(1) extra space. The idea is based on method 2 of this post.

```
1. Sort all element of array
2. Run loop from i=0 to n-2.
     Initialize two index variables l=i+1 and r=n-1
4. while (l < r)
     Check sum of arr[i], arr[l], arr[r] is
```

```
     zero or not if sum is zero then print the
     triplet and do l++ and r--.
5. If sum is less than zero then l++
6. If sum is greater than zero then r--
7. If not exist in array then print not found.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is zero
#include<bits/stdc++.h>
using namespace std;

// function to print triplets with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = false;

    // sort array elements
    sort(arr, arr+n);

    for (int i=0; i<n-1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
            if (x + arr[l] + arr[r] == 0)
            {
                // print elements if it's sum is zero
                printf("%d %d %d\n", x, arr[l], arr[r]);
                l++;
                r--;
                found = true;
            }

            // If sum of three elements is less
            // than zero then increment in left
            else if (x + arr[l] + arr[r] < 0)
                l++;

            // if sum is greater than zero than
            // decrement in right side
            else
                r--;
        }
```

```
    }

    if (found == false)
        cout << " No Triplet Found" << endl;
}

// Driven source
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}
```

Python3

```
 # python program to find triplets in a given
# array whose sum is zero

# function to print triplets with 0 sum
def findTriplets(arr, n):

    found = False

    # sort array elements
    arr.sort()

    for i in range(0, n-1):

        # initialize left and right
        l = i + 1
        r = n - 1
        x = arr[i]
        while (l < r):

            if (x + arr[l] + arr[r] == 0):
                # print elements if it's sum is zero
                print(x, arr[l], arr[r])
                l+=1
                r-=1
                found = True


            # If sum of three elements is less
            # than zero then increment in left
            elif (x + arr[l] + arr[r] < 0):
                l+=1
```

```
            # if sum is greater than zero than
            # decrement in right side
            else:
                r-=1

    if (found == False):
        print(" No Triplet Found")


# Driven source
arr = [0, -1, 2, -3, 1]
n = len(arr)
findTriplets(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

PHP

```
 <?php
// PHP program to find
// triplets in a given
// array whose sum is zero

// function to print
// triplets with 0 sum
function findTriplets($arr, $n)
{
    $found = false;

    // sort array elements
    sort($arr);

    for ($i = 0; $i < $n - 1; $i++)
    {
        // initialize left
        // and right
        $l = $i + 1;
        $r = $n - 1;
        $x = $arr[$i];
        while ($l < $r)
        {
            if ($x + $arr[$l] +
                     $arr[$r] == 0)
            {
                // print elements if
                // it's sum is zero
                echo $x," ", $arr[$l],
```

```
                        " ", $arr[$r], "\n";
                $l++;
                $r--;
                $found = true;
            }

            // If sum of three elements
            // is less than zero then
            // increment in left
            else if ($x + $arr[$l] +
                         $arr[$r] < 0)
                $l++;

            // if sum is greater than
            // zero than decrement
            // in right side
            else
                $r--;
        }
    }

    if ($found == false)
        echo " No Triplet Found" ,"\n";
}

// Driver Code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
-3 1 2
-1 0 1
```

Time Complexity : $O(n^2)$
Auxiliary Space : $O(1)$

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/find-triplets-array-whose-sum-equal-zero/

Chapter 84

Find all pairs (a, b) in an array such that a % b = k

Find all pairs (a, b) in an array such that a % b = k - GeeksforGeeks

Given an array with distinct elements, the task is to find the pairs in the array such that a % b = k, where k is a given integer.

Examples :

```
Input  :  arr[] = {2, 3, 5, 4, 7}
              k = 3
Output :  (7, 4), (3, 4), (3, 5), (3, 7)
7 % 4 = 3
3 % 4 = 3
3 % 5 = 3
3 % 7 = 3
```

A **Naive Solution** is to make all pairs one by one and check their modulo is equal to k or not. If equals to k, then print that pair.

C++

```
 // C++ implementation to find such pairs
#include <iostream>
using namespace std;

// Function to find pair such that (a % b = k)
bool printPairs(int arr[], int n, int k)
{
    bool isPairFound = true;
```

```
    // Consider each and every pair
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // Print if their modulo equals to k
            if (i != j && arr[i] % arr[j] == k) {
                cout << "(" << arr[i] << ", "
                     << arr[j] << ")"
                     << " ";
                isPairFound = true;
            }
        }
    }

    return isPairFound;
}

// Driver program
int main()
{
    int arr[] = { 2, 3, 5, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;

    if (printPairs(arr, n, k) == false)
        cout << "No such pair exists";

    return 0;
}
```

Java

```
 // Java implementation to find such pairs

class Test {
    // method to find pair such that (a % b = k)
    static boolean printPairs(int arr[], int n, int k)
    {
        boolean isPairFound = true;

        // Consider each and every pair
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // Print if their modulo equals to k
                if (i != j && arr[i] % arr[j] == k) {
                    System.out.print("(" + arr[i] + ", " + arr[j] + ")"
                                     + " ");
                    isPairFound = true;
```

```
                }
            }
        }

        return isPairFound;
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 2, 3, 5, 4, 7 };
        int k = 3;

        if (printPairs(arr, arr.length, k) == false)
            System.out.println("No such pair exists");
    }
}
```

Python3

```
 # Python3 implementation to find such pairs

# Function to find pair such that (a % b = k)
def printPairs(arr, n, k):

    isPairFound = True

    # Consider each and every pair
    for i in range(0, n):

        for j in range(0, n):

            # Print if their modulo equals to k
            if (i != j and arr[i] % arr[j] == k):

                print("(", arr[i], ", ", arr[j], ")",
                                sep = "", end = " ")
                isPairFound = True

    return isPairFound

# Driver Code
arr = [2, 3, 5, 4, 7]
n = len(arr)
k = 3
if (printPairs(arr, n, k) == False):
    print("No such pair exists")
```

```
# This article is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# implementation to find such pair
using System;

public class GFG {

    // method to find pair such that (a % b = k)
    static bool printPairs(int[] arr, int n, int k)
    {
        bool isPairFound = true;

        // Consider each and every pair
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
            {
                // Print if their modulo equals to k
                if (i != j && arr[i] % arr[j] == k)
                {
                    Console.Write("(" + arr[i] + ", "
                                + arr[j] + ")" + " ");
                    isPairFound = true;
                }
            }
        }

        return isPairFound;
    }

    // Driver method
    public static void Main()
    {
        int[] arr = { 2, 3, 5, 4, 7 };
        int k = 3;

        if (printPairs(arr, arr.Length, k) == false)
            Console.WriteLine("No such pair exists");
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
```

```
// PHP implementation to
// find such pairs

// Function to find pair
// such that (a % b = k)
function printPairs($arr, $n, $k)
{
    $isPairFound = true;

    // Consider each and every pair
    for ($i = 0; $i < $n; $i++)
    {
        for ( $j = 0; $j < $n; $j++)
        {
            // Print if their modulo
            // equals to k
            if ($i != $j && $arr[$i] %
                            $arr[$j] == $k)
            {
                echo "(" , $arr[$i] , ", ",
                       $arr[$j] , ")", " ";
                $isPairFound = true;
            }
        }
    }

    return $isPairFound;
}

// Driver Code
$arr = array(2, 3, 5, 4, 7);
$n = sizeof($arr);
$k = 3;

if (printPairs($arr, $n, $k) == false)
    echo "No such pair exists";

// This code is contributed by ajit
?>
```

Output :

```
(3, 5) (3, 4) (3, 7) (7, 4)
```

Time Complexity : $O(n^2)$

An **Efficient solution** is based on below observations :

1. If k itself is present in arr[], then k forms a pair with all elements arr[i] where k < arr[i]. For all such arr[i], we have k % arr[i] = k.
2. For all elements greater than or equal to arr[i], we use following fact.

```
If arr[i] % arr[j] = k,
==> arr[i] = x * arr[j] + k
==> (arr[i] - k) = x * arr[j]
We find all divisors of (arr[i] - k)
and see if they are present in arr[].
```

To quickly check if an element is present in array, we use hashing.

C++

```
// C++ program to to find all pairs such that
// a % b = k.
#include <bits/stdc++.h>
using namespace std;

// Utiltity function to find the divisors of
// n and store in vector v[]
vector<int> findDivisors(int n)
{
    vector<int> v;

    // Vector is used to store  the divisors
    for (int i = 1; i <= sqrt(n); i++) {
        if (n % i == 0) {
            // If n is a square number, push
            // only one occurrence
            if (n / i == i)
                v.push_back(i);
            else {
                v.push_back(i);
                v.push_back(n / i);
            }
        }
    }
    return v;
}

// Function to find pairs such that (a%b = k)
bool printPairs(int arr[], int n, int k)
{
    // Store all the elements in the map
```

```
    // to use map as hash for finding elements
    // in O(1) time.
    unordered_map<int, bool> occ;
    for (int i = 0; i < n; i++)
        occ[arr[i]] = true;

    bool isPairFound = false;
    for (int i = 0; i < n; i++) {
        // Print all the pairs with (a, b) as
        // (k, numbers greater than k) as
        // k % (num (> k)) = k i.e. 2%4 = 2
        if (occ[k] && k < arr[i]) {
            cout << "(" << k << ", " << arr[i] << ") ";
            isPairFound = true;
        }

        // Now check for the current element as 'a'
        // how many b exists such that a%b = k
        if (arr[i] >= k) {
            // find all the divisors of (arr[i]-k)
            vector<int> v = findDivisors(arr[i] - k);

            // Check for each divisor i.e. arr[i] % b = k
            // or not, if yes then print that pair.
            for (int j = 0; j < v.size(); j++) {
                if (arr[i] % v[j] == k && arr[i] != v[j] && occ[v[j]]) {
                    cout << "(" << arr[i] << ", "
                         << v[j] << ") ";
                    isPairFound = true;
                }
            }

            // Clear vector
            v.clear();
        }
    }

    return isPairFound;
}

// Driver program
int main()
{
    int arr[] = { 3, 1, 2, 5, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;

    if (printPairs(arr, n, k) == false)
```

```
        cout << "No such pair exists";
    return 0;
}
```

Java

```
 // Java program to to find all pairs such that
// a % b = k.

import java.util.HashMap;
import java.util.Vector;

class Test {
    // Utility method to find the divisors of
    // n and store in vector v[]
    static Vector<Integer> findDivisors(int n)
    {
        Vector<Integer> v = new Vector<>();

        // Vector is used to store  the divisors
        for (int i = 1; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                // If n is a square number, push
                // only one occurrence
                if (n / i == i)
                    v.add(i);
                else {
                    v.add(i);
                    v.add(n / i);
                }
            }
        }
        return v;
    }

    // method to find pairs such that (a%b = k)
    static boolean printPairs(int arr[], int n, int k)
    {
        // Store all the elements in the map
        // to use map as hash for finding elements
        // in O(1) time.
        HashMap<Integer, Boolean> occ = new HashMap<>();
        for (int i = 0; i < n; i++)
            occ.put(arr[i], true);

        boolean isPairFound = false;
        for (int i = 0; i < n; i++) {
            // Print all the pairs with (a, b) as
```

```
            // (k, numbers greater than k) as
            // k % (num (> k)) = k i.e. 2%4 = 2
            if (occ.get(k) && k < arr[i]) {
                System.out.print("(" + k + ", " + arr[i] + ") ");
                isPairFound = true;
            }

            // Now check for the current element as 'a'
            // how many b exists such that a%b = k
            if (arr[i] >= k) {
                // find all the divisors of (arr[i]-k)
                Vector<Integer> v = findDivisors(arr[i] - k);

                // Check for each divisor i.e. arr[i] % b = k
                // or not, if yes then print that pair.
                for (int j = 0; j < v.size(); j++) {
                    if (arr[i] % v.get(j) == k && arr[i] != v.get(j) && occ.get(v.get(j))) {
                        System.out.print("(" + arr[i] + ", "
                                         + v.get(j) + ") ");
                        isPairFound = true;
                    }
                }

                // Clear vector
                v.clear();
            }
        }

        return isPairFound;
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 3, 1, 2, 5, 4 };
        int k = 2;

        if (printPairs(arr, arr.length, k) == false)
            System.out.println("No such pair exists");
    }
}
```

Output:

```
(2, 3) (2, 5) (5, 3) (2, 4)
```

Time Complexity: O(n* sqrt(max)) where max is the maximum element in the array.

Reference:
https://stackoverflow.com/questions/12732939/find-pairs-in-an-array-such-that-ab-k-where-k-is-a-given-integer

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/find-pairs-b-array-b-k/

Chapter 85

Find all pairs (a,b) and (c,d) in array which satisfy ab = cd

Find all pairs (a,b) and (c,d) in array which satisfy ab = cd - GeeksforGeeks

Given an array of distinct integers, the task is to find two pairs (a, b) and (c, d) such that ab = cd, where a, b, c and d are distinct elements.

Examples:

```
Input  : arr[] = {3, 4, 7, 1, 2, 9, 8}
Output : 4 2 and 1 8
Product of 4 and 2 is 8 and
also product of 1 and 8 is 8 .

Input  : arr[] = {1, 6, 3, 9, 2, 10};
Output : 6 3 and 9 2
```

A **Simple Solution** is to run four loops to generate all possible quadruples of array element. For every quadruple (a, b, c, d), check if a*b = c*d. Time complexity of this solution is $O(n^4)$.

An **Efficient Solution** of this problem is to use hashing. We use product as key and pair as value in hash table.

```
1. For i=0 to n-1
2.   For j=i+1 to n-1
       a) Find  prod = arr[i]*arr[j]
       b) If prod is not available in hash then make
            H[prod] = make_pair(i, j) // H is hash table
       c) If product is also available in hash
```

```
          then print previous and current elements
          of array
```

C++

```
 // C++ program to find four elements a, b, c
// and d in array such that ab = cd
#include<bits/stdc++.h>
using namespace std;

// Function to find out four elements in array
// whose product is ab = cd
void findPairs(int arr[], int n)
{
    bool found = false;
    unordered_map<int, pair < int, int > > H;
    for (int i=0; i<n; i++)
    {
        for (int j=i+1; j<n; j++)
        {
            // If product of pair is not in hash table,
            // then store it
            int prod = arr[i]*arr[j];
            if (H.find(prod) == H.end())
                H[prod] = make_pair(i,j);

            // If product of pair is also available in
            // then print current and previous pair
            else
            {
                pair<int,int> pp = H[prod];
                cout << arr[pp.first] << " " << arr[pp.second]
                     << " and " << arr[i]<<" "<<arr[j]<<endl;
                found = true;
            }
        }
    }
    // If no pair find then print not found
    if (found == false)
        cout << "No pairs Found" << endl;
}

//Driven code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int n = sizeof(arr)/sizeof(int);
    findPairs(arr, n);
```

```
    return 0;
}
```

Java

```
 // Java program to find four elements a, b, c
// and d in array such that ab = cd
import java.io.*;
import java.util.*;

class GFG {

    public static class pair {

        int first,second;

        pair(int f, int s)
        {
            first = f;
            second = s;
        }
    };

    // Function to find out four elements
    // in array whose product is ab = cd
    public static void findPairs(int arr[], int n)
    {

        boolean found = false;
        HashMap<Integer, pair> hp =
                     new HashMap<Integer, pair>();

        for(int i = 0; i < n; i++)
        {
            for(int j = i + 1; j < n; j++)
            {

                // If product of pair is not in
                // hash table, then store it
                int prod = arr[i] * arr[j];

                if(!hp.containsKey(prod))
                    hp.put(prod, new pair(i,j));

                // If product of pair is also
                // available in then print
                // current and previous pair
                else
```

```
                {
                    pair p = hp.get(prod);
                    System.out.println(arr[p.first]
                              + " " + arr[p.second]
                              + " " + "and" + " " +
                             arr[i] + " " + arr[j]);
                    found = true;
                }
            }
        }

        // If no pair find then print not found
        if(found == false)
        System.out.println("No pairs Found");
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int n = arr.length;
        findPairs(arr, n);
    }
}

// This code is contributed by akash1295.
```

Output:

```
1 6 and 2 3
1 8 and 2 4
2 6 and 3 4
3 8 and 4 6
```

Time Complexity : O(n^2) assuming hash search and insert operations take O(1) time.

Related Article:
Find four elements a, b, c and d in an array such that a+b = c+d

Improved By : akash1295

Source

https://www.geeksforgeeks.org/find-pairs-ab-cd-array-satisfy-ab-cd/

Chapter 86

Find all permuted rows of a given row in a matrix

Find all permuted rows of a given row in a matrix - GeeksforGeeks

We are given a m*n matrix of positive integers and a row number. The task is to find all rows in given matrix which are permutations of given row elements. It is also given that values in every row are distinct.

Examples:

```
Input : mat[][] = {{3, 1, 4, 2},
                   {1, 6, 9, 3},
                   {1, 2, 3, 4},
                   {4, 3, 2, 1}}
        row = 3
Output: 0, 2
Rows at indexes 0 and 2 are permutations of
row at index 3.
```

A **simple solution** is to one by one sort all rows and check all rows. If any row is completely equal to given row that means current row is a permutation of given row. Time complexity for this approach will be O(m*n log n).

An **efficient approach** is to use a hashing. Simply create a hash set for given row. After hash set creation, traverse through remaining rows and for every row check if all of its elements are present in hash set or not.

CPP

```
 // C++ program to find all permutations of a given row
#include<bits/stdc++.h>
```

```
#define MAX 100

using namespace std;

// Function to find all permuted rows of a given row r
void permutatedRows(int mat[][MAX], int m, int n, int r)
{
    // Creating an empty set
    unordered_set<int> s;

    // Count frequencies of elements in given row r
    for (int j=0; j<n; j++)
        s.insert(mat[r][j]);

    // Traverse through all remaining rows
    for (int i=0; i<m; i++)
    {
        // we do not need to check for given row r
        if (i==r)
            continue;

        // initialize hash i.e; count frequencies
        // of elements in row i
        int j;
        for (j=0; j<n; j++)
            if (s.find(mat[i][j]) == s.end())
                break;
        if (j != n)
           continue;

        cout << i << ", ";
    }
}

// Driver program to run the case
int main()
{
    int m = 4, n = 4,r = 3;
    int mat[][MAX] = {{3, 1, 4, 2},
                      {1, 6, 9, 3},
                      {1, 2, 3, 4},
                      {4, 3, 2, 1}};
    permutatedRows(mat, m, n, r);
    return 0;
}
```

Python3

```
 # Python program to find all
# permutations of a given row

# Function to find all
# permuted rows of a given row r
def permutatedRows(mat, m, n, r):


    # Creating an empty set
    s=set()

    # Count frequencies of
    # elements in given row r
    for j in range(n):
        s.add(mat[r][j])

    # Traverse through all remaining rows
    for i in range(m):

        # we do not need to check
        # for given row r
        if i == r:
            continue

        # initialize hash i.e
        # count frequencies
        # of elements in row i
        for j in range(n):
            if mat[i][j] not in s:

                # to avoid the case when last
                # element does not match
                j = j - 2
                break;
        if j + 1 != n:
            continue
        print(i)



# Driver program to run the case
m = 4
n = 4
r = 3
mat = [[3, 1, 4, 2],
       [1, 6, 9, 3],
       [1, 2, 3, 4],
       [4, 3, 2, 1]]
```

```
permutatedRows(mat, m, n, r)

# This code is contributed
# by Upendra Singh Bartwal.
```

Output:

```
0, 2
```

Time complexity : O(m*n)
Auxiliary space : O(n)

Another approach to the solution using Standard Template Library(STL):

CPP

```
 // C++ program to find all permutations of a given row
#include<bits/stdc++.h>
#define MAX 100

using namespace std;

// Function to find all permuted rows of a given row r
void permutatedRows(int mat[][MAX], int m, int n, int r)
{
   for (int i=0; i<m&&i!=r; i++){
        if(is_permutation(mat[i],mat[i]+n,mat[r])) cout<<i<<",";
    }
}

// Driver program to run the case
int main()
{
    int m = 4, n = 4,r = 3;
    int mat[][MAX] = {{3, 1, 4, 2},
                      {1, 6, 9, 3},
                      {1, 2, 3, 4},
                      {4, 3, 2, 1}};
    permutatedRows(mat, m, n, r);
    return 0;
}
```

Output:

```
0, 2
```

Exercise :
Extend the above solution to work for input matrix where all elements of a row don't have be distinct. (Hit : We can use Hash Map instead of Hash Set)

Improved By : banavath santhosh

Source

https://www.geeksforgeeks.org/find-permuted-rows-given-row-matrix/

Chapter 87

Find all strings that match specific pattern in a dictionary

Find all strings that match specific pattern in a dictionary - GeeksforGeeks

Given a dictionary of words, find all strings that matches the given pattern where every character in the pattern is uniquely mapped to a character in the dictionary.

Examples:

```
Input:
dict = ["abb", "abc", "xyz", "xyy"];
pattern = "foo"
Output: [xyy abb]
Explanation:
xyy and abb have same character at index 1 and 2 like the pattern

Input:
dict = ["abb", "abc", "xyz", "xyy"];
pat = "mno"
Output: [abc xyz]
Explanation:
abc and xyz have all distinct characters, similar to the pattern

Input:
dict = ["abb", "abc", "xyz", "xyy"];
pattern = "aba"
Output: []
Explanation:
Pattern has same character at index 0 and 2.
No word in dictionary follows the pattern.
```

```
Input:
dict = ["abab", "aba", "xyz", "xyx"];
pattern = "aba"
Output: [aba xyx]
Explanation:
aba and xyx have same character at index 0 and 2 like the pattern
```

The idea is to encode the pattern in such a way that any word from the dictionary that matches the pattern will have same hash as that of the pattern after encoding. We iterate through all words in dictionary one by one and print the words that have same hash as that of the pattern.

Below is C++ implementation of above idea –

```
 // C++ program to print all the strings that match the
// given pattern where every character in the pattern is
// uniquely mapped to a character in the dictionary
#include <bits/stdc++.h>
using namespace std;

// Function to encode given string
string encodeString(string str)
{
    unordered_map<char, int> map;
    string res = "";
    int i = 0;

    // for each character in given string
    for (char ch : str)
    {
        // If the character is occurring for the first
        // time, assign next unique number to that char
        if (map.find(ch) == map.end())
            map[ch] = i++;

        // append the number associated with current
        // character into the output string
        res += to_string(map[ch]);
    }

    return res;
}

// Function to print all the strings that match the
// given pattern where every character in the pattern is
// uniquely mapped to a character in the dictionary
void findMatchedWords(unordered_set<string> dict,
                      string pattern)
```

```
{
    // len is length of the pattern
    int len = pattern.length();

    // encode the string
    string hash = encodeString(pattern);

    // for each word in the dictionary
    for (string word : dict)
    {
        // If size of pattern is same as size of current
        // dictionary word and both pattern and the word
        // has same hash, print the word
        if (word.length() == len && encodeString(word) == hash)
            cout << word << " " ;
    }
}

// Driver code
int main()
{
    unordered_set<string> dict = { "abb", "abc", "xyz", "xyy" };
    string pattern = "foo";

    findMatchedWords(dict, pattern);

    return 0;
}
```

Output:

```
xyy abb
```

Source

https://www.geeksforgeeks.org/find-all-strings-that-match-specific-pattern-in-a-dictionary/

Chapter 88

Find any one of the multiple repeating elements in read only array

Find any one of the multiple repeating elements in read only array - GeeksforGeeks

Given a **read only** array of size (n+1), find one of the multiple repeating elements in the array where the array contains integers only between 1 and n.
Read only array means that the contents of the array can't be modified.

Examples:

```
Input : n = 5
        arr[] = {1, 1, 2, 3, 5, 4}
Output : One of the numbers repeated in the array is: 1

Input : n = 10
        arr[] = {10, 1, 2, 3, 5, 4, 9, 8, 5, 6, 4}
Output : One of the numbers repeated in the array is: 4 OR 5
```

Since, the size of the array is n+1 and elements ranges from 1 to n then it is confirmed that there will be at least one repeating element.

A **simple solution** is to create a count array and store counts of all elements. As soon as we encounter an element with count more than 1, we return it. This solution works in O(n) time and requires O(n) extra space.

A **space optimized solution** is to break the given range (from 1 to n) into blocks of size equal to sqrt(n). We maintain the count of elements belonging to each block for every block. Now as the size of array is (n+1) and blocks are of size sqrt(n), then there will be one such block whose size will be more than sqrt(n). For the block whose count is greater than

sqrt(n), we can use hashing for the elements of this block to find which element appears more than once.

Explanation:
The method described above works because of the following two reasons:

1. There would always be a block which has count greater than sqrt(n) because of one extra element. Even when one extra element has been added it will occupy a position in one of the blocks only, making that block to be selected.
2. The selected block definitely has a repeating element. Consider that i^{th} block is selected. Size of the block is greater than sqrt(n) (Hence, it is selected) Maximum distinct elements in this block = sqrt(n). Thus, size can be greater than sqrt(n) only if there is a repeating element in range (i*sqrt(n), (i+1)*sqrt(n)].

Note: The last block formed may or may not have range equal to sqrt(n). Thus, checking if this block has a repeating element will be different than other blocks. However, this difficulty can be overcome from implementation point of view by initialising the selected block with the last block. This is safe because at least one block has to get selected.

Below is the step by step algorithm to solve this problem:

1. Divide the array in blocks of size sqrt(n).
2. Make a count array which stores the count of element for each block.
3. Pick up the block which has count more than sqrt(n), setting the last block as default.
4. For the elements belonging to the selected block, use the method of hashing(explained in next step) to find the repeating element in that block.
5. We can create a hash array of key value pair, where key is the element in the block and value is the count of number of times the given key is appearing. This can be easily implemented using unordered_map in C++ STL.

Below is C++ implementation of above idea:

```
 // C++ program to find one of the repeating
// elements in a read only array
#include <iostream>
#include <cmath>
#include <unordered_map>
using namespace std;

// Function to find one of the repeating
// elements
int findRepeatingNumber(const int arr[], int n)
{
    // Size of blocks except the
    // last block is sq
    int sq = sqrt(n);
```

```
// Number of blocks to incorporate 1 to
// n values blocks are numbered from 0
// to range-1 (both included)
int range = (n / sq) + 1;

// Count array maintains the count for
// all blocks
int count[range] = {0};

// Traversing the read only array and
// updating count
for (int i = 0; i <= n; i++)
{
    // arr[i] belongs to block number
    // (arr[i]-1)/sq i is considered
    // to start from 0
    count[(arr[i] - 1) / sq]++;
}

// The selected_block is set to last
// block by default. Rest of the blocks
// are checked
int selected_block = range - 1;
for (int i = 0; i < range - 1; i++)
{
    if (count[i] > sq)
    {
        selected_block = i;
        break;
    }
}

// after finding block with size > sq
// method of hashing is used to find
// the element repeating in this block
unordered_map<int, int> m;
for (int i = 0; i <= n; i++)
{
    // checks if the element belongs to the
    // selected_block
    if ( ((selected_block * sq) < arr[i]) &&
            (arr[i] <= ((selected_block + 1) * sq)))
    {
        m[arr[i]]++;

        // repeating element found
        if (m[arr[i]] > 1)
            return arr[i];
```

```
        }
    }

    // return -1 if no repeating element exists
    return -1;
}

// Driver Program
int main()
{
    // read only array, not to be modified
    const int arr[] = { 1, 1, 2, 3, 5, 4 };

    // array of size 6(n + 1) having
    // elements between 1 and 5
    int n = 5;

    cout << "One of the numbers repeated in"
         " the array is: "
         << findRepeatingNumber(arr, n) << endl;
}
```

Output:

```
One of the numbers repeated in the array is: 1
```

Time Complexity: O(N)
Auxiliary Space: sqrt(N)

Source

https://www.geeksforgeeks.org/find-one-multiple-repeating-elements-read-array/

Chapter 89

Find common elements in three linked lists

Find common elements in three linked lists - GeeksforGeeks

Given three linked lists, find all common element among the three linked lists.

Examples:

```
Input :
   10 15 20 25 12
   10 12 13 15
   10 12 15 24 25 26
Output : 10 12 15

Input :
   1 2 3 4 5
   1 2 3 4 6 9 8
   1 2 4 5 10
Output : 1 2 4
```

Method 1 : (Simple)
Use three-pointers to iterate the given three linked lists and if any element common print that element.
Time complexity of the above solution will be O(N*N*N)

Method 2 : (Use Merge Sort)
In this method, we first sort the three lists and then we traverse the sorted lists to get the intersection.

Following are the steps to be followed to get intersection of three lists:

1) Sort the first Linked List using merge sort. This step takes O(mLogm) time. Refer this post for details of this step.

2) Sort the second Linked List using merge sort. This step takes O(nLogn) time. Refer this post for details of this step.

3) Sort the third Linked List using merge sort. This step takes O(pLogp) time. Refer this post for details of this step.

3) Linearly scan three sorted lists to get the intersection. This step takes O(m + n + p) time. This step can be implemented using the same algorithm as sorted arrays algorithm discussed here.

Time complexity of this method is O(mLogm + nLogn + plogp) which is better than method 1's time complexity.

Method 3 : (Hashing)
Following are the steps to be followed to get intersection of three lists using hashing:
1) Create an empty hash table. Iterate through the first linked list and mark all the element frequency as 1 in the hash table. This step takes O(m) time.
2) Iterate through the second linked list and if current element frequency is 1 in hash table mark it as 2. This step takes O(n) time.
3) Iterate the third linked list and if the current element frequency is 2 in hash table mark it as 3. This step takes O(p) time.
4) Now iterate first linked list again to check the frequency of elements. if an element with frequency three exist in hash table, it will be present in the intersection of three linked lists. This step takes O(m) time.

Time complexity of this method is O(m + n + p) which is better than time complexity of method 1 and 2.

Below is the C++ implementation of the above idea.

```
 // C++ program to find common element
// in three unsorted linked list
#include <bits/stdc++.h>
#define max 1000000
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the
beginning of a linked list */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
            (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
```

```
}

/* print the common element in between
given three linked list*/
void Common(struct Node* head1,
            struct Node* head2, struct Node* head3)
{

    // Creating empty hash table;
    unordered_map<int, int> hash;

    struct Node* p = head1;
    while (p != NULL) {

        // set frequency by 1
        hash[p->data] = 1;
        p = p->next;
    }

    struct Node* q = head2;
    while (q != NULL) {

        // if the element is already exist in the
        // linked list set its frequency 2
        if (hash.find(q->data) != hash.end())
            hash[q->data] = 2;
        q = q->next;
    }

    struct Node* r = head3;
    while (r != NULL) {
        if (hash.find(r->data) != hash.end() &&
            hash[r->data] == 2)

        // if the element frquancy is 2 it means
        // its present in both the first and second
        // linked list set its frquancy 3
        hash[r->data] = 3;
        r = r->next;
    }


    for (auto x : hash) {

        // if current frequency is 3 its means
        // element is common in all the given
        // linked list
        if (x.second == 3)
```

```
            cout << x.first << " ";
    }
}

// Driver code
int main()
{

    // first list
    struct Node* head1 = NULL;
    push(&head1, 20);
    push(&head1, 5);
    push(&head1, 15);
    push(&head1, 10);

    // second list
    struct Node* head2 = NULL;
    push(&head2, 10);
    push(&head2, 20);
    push(&head2, 15);
    push(&head2, 8);

    // third list
    struct Node* head3 = NULL;
    push(&head3, 10);
    push(&head3, 2);
    push(&head3, 15);
    push(&head3, 20);

    Common(head1, head2, head3);

    return 0;
}
```

Output:

```
10 15 20
```

Time Complexity : O(m + n + p)

Source

https://www.geeksforgeeks.org/find-common-elements-in-three-linked-lists/

Chapter 90

Find distinct elements common to all rows of a matrix

Find distinct elements common to all rows of a matrix - GeeksforGeeks

Given a n x n matrix. The problem is to find all the distinct elements common to all rows of the matrix. The elements can be printed in any order.

Examples:

```
Input : mat[][] = {  {2, 1, 4, 3},
                     {1, 2, 3, 2},
                     {3, 6, 2, 3},
                     {5, 2, 5, 3}  }
Output : 2 3

Input : mat[][] = {  {12, 1, 14, 3, 16},
                     {14, 2, 1, 3, 35},
                     {14, 1, 14, 3, 11},
                     {14, 25, 3, 2, 1},
                     {1, 18, 3, 21, 14}  }
Output : 1 3 14
```

Method 1: Using three nested loops. Check if an element of 1st row is present in all the subsequent rows. Time Complexity of $O(n^3)$. Extra space could be required to handle the duplicate elements.

Method 2: Sort all the rows of the matrix individually in increasing order. Then apply a modified approach of the problem of finding common elements in 3 sorted arrays. Below an implementation for the same is given.

C++

```
 // C++ implementation to find distinct elements
// common to all rows of a matrix
#include <bits/stdc++.h>
using namespace std;
const int MAX = 100;

// function to individually sort
// each row in increasing order
void sortRows(int mat[][MAX], int n)
{
    for (int i=0; i<n; i++)
        sort(mat[i], mat[i] + n);
}

// function to find all the common elements
void findAndPrintCommonElements(int mat[][MAX], int n)
{
    // sort rows individually
    sortRows(mat, n);

    // current column index of each row is stored
    // from where the element is being searched in
    // that row
    int curr_index[n];
    memset(curr_index, 0, sizeof(curr_index));
    int f = 0;

    for (; curr_index[0]<n; curr_index[0]++)
    {
        // value present at the current column index
        // of 1st row
        int value = mat[0][curr_index[0]];

        bool present = true;

        // 'value' is being searched in all the
        // subsequent rows
        for (int i=1; i<n; i++)
        {
            // iterate through all the elements of
            // the row from its current column index
            // till an element greater than the 'value'
            // is found or the end of the row is
            // encountered
            while (curr_index[i] < n &&
                   mat[i][curr_index[i]] <= value)
```

```
                curr_index[i]++;

            // if the element was not present at the column
            // before to the 'curr_index' of the row
            if (mat[i][curr_index[i]-1] != value)
                present = false;

            // if all elements of the row have
            // been traversed
            if (curr_index[i] == n)
            {
                f = 1;
                break;
            }
        }

        // if the 'value' is common to all the rows
        if (present)
            cout << value << " ";

        // if any row have been completely traversed
        // then no more common elements can be found
        if (f == 1)
            break;
    }
}

// Driver program to test above
int main()
{
    int mat[][MAX] = {  {12, 1, 14, 3, 16},
        {14, 2, 1, 3, 35},
        {14, 1, 14, 3, 11},
        {14, 25, 3, 2, 1},
        {1, 18, 3, 21, 14}
    };

    int n = 5;
    findAndPrintCommonElements(mat, n);
    return 0;
}
```

Java

```
 // JAVA Code to find distinct elements
// common to all rows of a matrix
import java.util.*;
```

```
class GFG {

    // function to individually sort
    // each row in increasing order
    public static void sortRows(int mat[][], int n)
    {
        for (int i=0; i<n; i++)
            Arrays.sort(mat[i]);
    }

    // function to find all the common elements
    public static void findAndPrintCommonElements(int mat[][],
                                                   int n)
    {
        // sort rows individually
        sortRows(mat, n);

        // current column index of each row is stored
        // from where the element is being searched in
        // that row
        int curr_index[] = new int[n];

        int f = 0;

        for (; curr_index[0]<n; curr_index[0]++)
        {
            // value present at the current column index
            // of 1st row
            int value = mat[0][curr_index[0]];

            boolean present = true;

            // 'value' is being searched in all the
            // subsequent rows
            for (int i=1; i<n; i++)
            {
                // iterate through all the elements of
                // the row from its current column index
                // till an element greater than the 'value'
                // is found or the end of the row is
                // encountered
                while (curr_index[i] < n &&
                       mat[i][curr_index[i]] <= value)
                    curr_index[i]++;

                // if the element was not present at the
                // column before to the 'curr_index' of the
               // row
```

```
                if (mat[i][curr_index[i]-1] != value)
                    present = false;

                // if all elements of the row have
                // been traversed
                if (curr_index[i] == n)
                {
                    f = 1;
                    break;
                }
            }

            // if the 'value' is common to all the rows
            if (present)
               System.out.print(value+" ");

            // if any row have been completely traversed
            // then no more common elements can be found
            if (f == 1)
                break;
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int mat[][] = {  {12, 1, 14, 3, 16},
                         {14, 2, 1, 3, 35},
                         {14, 1, 14, 3, 11},
                         {14, 25, 3, 2, 1},
                         {1, 18, 3, 21, 14}
                                           };

            int n = 5;
            findAndPrintCommonElements(mat, n);
    }
  }
// This code is contributed by Arnav Kr. Mandal.
```

Output:

```
1 3 14
```

Time Complexity: O(n^2log n), each row of size n requires O(nlogn) for sorting and there are total n rows.
Auxiliary Space : O(n) to store current column indexes for each row.

Method 3: It uses the concept of hashing. The following steps are:

1. Map the element of 1st row in a hash table. Let it be **hash**.
2. Fow row = 2 to n
3. Map each element of the current row into a temporary hash table. Let it be **temp**.
4. Iterate through the elements of **hash** and check that the elements in **hash** are present in **temp**. If not present then delete those elements from **hash**.
5. When all the rows are being processed in this manner, then the elements left in **hash** are the required common elements.

```
 // C++ program to find distinct elements
// common to all rows of a matrix
#include <bits/stdc++.h>
using namespace std;

const int MAX = 100;

// function to individually sort
// each row in increasing order
void findAndPrintCommonElements(int mat[][MAX], int n)
{
    unordered_set<int> us;

    // map elements of first row
    // into 'us'
    for (int i=0; i<n; i++)
        us.insert(mat[0][i]);

    for (int i=1; i<n; i++)
    {
        unordered_set<int> temp;
        // mapping elements of current row
        // in 'temp'
        for (int j=0; j<n; j++)
            temp.insert(mat[i][j]);

        unordered_set<int>:: iterator itr;

        // iterate through all the elements
        // of 'us'
        for (itr=us.begin(); itr!=us.end(); itr++)

            // if an element of 'us' is not present
            // into 'temp', then erase that element
            // from 'us'
            if (temp.find(*itr) == temp.end())
```

```
                us.erase(*itr);

        // if size of 'us' becomes 0,
        // then there are no common elements
        if (us.size() == 0)
            break;
    }

    // print the common elements
    unordered_set<int>:: iterator itr;
    for (itr=us.begin(); itr!=us.end(); itr++)
        cout << *itr << " ";
}

// Driver program to test above
int main()
{
    int mat[][MAX] = { {2, 1, 4, 3},
                       {1, 2, 3, 2},
                       {3, 6, 2, 3},
                       {5, 2, 5, 3}  };
    int n = 4;
    findAndPrintCommonElements(mat, n);
    return 0;
}
```

Output:

```
3 2
```

Time Complexity: O(n^2)
Space Complexity: O(n)

Source

https://www.geeksforgeeks.org/find-distinct-elements-common-rows-matrix/

Chapter 91

Find duplicates in a given array when elements are not limited to a range

Find duplicates in a given array when elements are not limited to a range - GeeksforGeeks

Given an array of n integers. The task is to print the duplicates in the given array. If there are no duplicates then print -1.

Examples:

```
Input : {2, 10, 100, 2, 10, 11}
Output : 2 10

Input : {5, 40, 1, 40, 100000, 1, 5, 1}
Output : 5 40 1
```

Note: The duplicate elements can be printed in any order.

Simple Approach: By using two loops. It has a time complexity of $O(n^2)$.

Efficient Approach: Use unordered_map for hashing. Count frequency of occurrence of each element and the elements with frequency more than 1 is printed. **unordered_map** is used as range of integers is not known.

C++

```
 // C++ program to find
// duplicates in the given array
#include <bits/stdc++.h>
using namespace std;
```

```
// function to find and print duplicates
void printDuplicates(int arr[], int n)
{
    // unordered_map to store frequencies
    unordered_map<int, int> freq;
    for (int i=0; i<n; i++)
        freq[arr[i]]++;

    bool dup = false;
    unordered_map<int, int>:: iterator itr;
    for (itr=freq.begin(); itr!=freq.end(); itr++)
    {
        // if frequency is more than 1
        // print the element
        if (itr->second > 1)
        {
            cout << itr->first << " ";
            dup = true;
        }
    }

    // no duplicates present
    if (dup == false)
        cout << "-1";
}

// Driver program to test above
int main()
{
    int arr[] = {12, 11, 40, 12, 5, 6, 5, 12, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    printDuplicates(arr, n);
    return 0;
}
```

Java

```
 // Java program to find
// duplicates in the given array
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class FindDuplicatedInArray
{
    // Driver program
    public static void main(String[] args)
    {
```

```
        int arr[] = {12, 11, 40, 12, 5, 6, 5, 12, 11};
        int n = arr.length;
        printDuplicates(arr, n);
    }
    // function to find and print duplicates
    private static void printDuplicates(int[] arr, int n)
    {
        Map<Integer,Integer> map = new HashMap<>();
        int count = 0;
        boolean dup = false;
        for(int i = 0; i < n; i++){
            if(map.containsKey(arr[i])){
                count = map.get(arr[i]);
                map.put(arr[i], count + 1);
            }
            else{
                map.put(arr[i], 1);
            }
        }

        for(Entry<Integer,Integer> entry : map.entrySet())
        {
            // if frequency is more than 1
            // print the element
            if(entry.getValue() > 1){
                System.out.print(entry.getKey()+ " ");
                dup = true;
            }
        }
        // no duplicates present
        if(!dup){
            System.out.println("-1");
        }
    }
}
```

Output:

```
12 11 5
```

Time Complexity: O(n)

Related Post :
Print All Distinct Elements of a given integer array
Find duplicates in O(n) time and O(1) extra space | Set 1
Duplicates in an array in O(n) and by using O(1) extra space | Set-2
Print all the duplicates in the input string

Improved By : WalterFrobin

Source

https://www.geeksforgeeks.org/find-duplicates-given-array-elements-not-limited-range/

Chapter 92

Find elements which are present in first array and not in second

Find elements which are present in first array and not in second - GeeksforGeeks

Given two arrays, the task is that we find numbers which are present in first array, but not present in the second array.
Examples :

```
Input : a[] = {1, 2, 3, 4, 5, 10};
    b[] = {2, 3, 1, 0, 5};
Output : 4 10
4 and 10 are present in first array, but
not in second array.

Input : a[] = {4, 3, 5, 9, 11};
        b[] = {4, 9, 3, 11, 10};
Output : 5
```

Method 1 (Simple)
A Naive Approach is to use two loops and check element which not present in second array.

C++

```
 // C++ simple program to
// find elements which are
// not present in second array
#include<bits/stdc++.h>
using namespace std;

// Function for finding
```

```
// elements which are there
// in a[]  but not in b[].
void findMissing(int a[], int b[],
                 int n, int m)
{
    for (int i = 0; i < n; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (a[i] == b[j])
                break;

        if (j == m)
            cout << a[i] << " ";
    }
}

// Driver code
int main()
{
    int a[] = { 1, 2, 6, 3, 4, 5 };
    int b[] = { 2, 4, 3, 1, 0 };
    int n = sizeof(a) / sizeof(a[0]);
    int m = sizeof(b) / sizeof(b[1]);
    findMissing(a, b, n, m);
    return 0;
}
```

Java

```
 // Java simple program to
// find elements which are
// not present in second array
class GFG
{

    // Function for finding elements
    // which are there in a[] but not
    // in b[].
    static void findMissing(int a[], int b[],
                            int n, int m)
    {
        for (int i = 0; i < n; i++)
        {
            int j;

            for (j = 0; j < m; j++)
                if (a[i] == b[j])
```

```
                    break;

            if (j == m)
                System.out.print(a[i] + " ");
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        int a[] = { 1, 2, 6, 3, 4, 5 };
        int b[] = { 2, 4, 3, 1, 0 };

        int n = a.length;
        int m = b.length;

        findMissing(a, b, n, m);
    }
}

// This code is contributed
// by Anant Agarwal.
```

C#

```
 // C# simple program to find elements
// which are not present in second array
using System;

class GFG {

    // Function for finding elements
    // which are there in a[] but not
    // in b[].
    static void findMissing(int []a, int []b,
                            int n, int m)
    {
        for (int i = 0; i < n; i++)
        {
            int j;

            for (j = 0; j < m; j++)
                if (a[i] == b[j])
                    break;

            if (j == m)
                Console.Write(a[i] + " ");
        }
```

```
    }

    // Driver code
    public static void Main()
    {
        int []a = {1, 2, 6, 3, 4, 5};
        int []b = {2, 4, 3, 1, 0};

        int n = a.Length;
        int m = b.Length;

        findMissing(a, b, n, m);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP simple program to find
// elements which are not
// present in second array

// Function for finding
// elements which are there
// in a[] but not in b[].
function findMissing( $a, $b, $n, $m)
{
    for ( $i = 0; $i < $n; $i++)
    {
        $j;
        for ($j = 0; $j < $m; $j++)
            if ($a[$i] == $b[$j])
                break;

        if ($j == $m)
            echo $a[$i] , " ";
    }
}

// Driver code
$a = array( 1, 2, 6, 3, 4, 5 );
$b = array( 2, 4, 3, 1, 0 );
$n = count($a);
$m = count($b);
findMissing($a, $b, $n, $m);
```

```
// This code is contributed by anuj_67.
?>
```

Output :

```
6 5
```

Method 2 (Use Hashing)
In this method, we store all elements of second array in a hash table (unordered_set). One by one check all elements of first array and print all those elements which are not present in the hash table.

C++

```
 // C++ efficient program to
// find elements which are not
// present in second array
#include<bits/stdc++.h>
using namespace std;

// Function for finding
// elements which are there
// in a[] but not in b[].
void findMissing(int a[], int b[],
                 int n, int m)
{
    // Store all elements of
    // second array in a hash table
    unordered_set <int> s;
    for (int i = 0; i < m; i++)
        s.insert(b[i]);

    // Print all elements of
    // first array that are not
    // present in hash table
    for (int i = 0; i < n; i++)
        if (s.find(a[i]) == s.end())
            cout << a[i] << " ";
}

// Driver code
int main()
{
    int a[] = { 1, 2, 6, 3, 4, 5 };
    int b[] = { 2, 4, 3, 1, 0 };
```

```
    int n = sizeof(a) / sizeof(a[0]);
    int m = sizeof(b) / sizeof(b[1]);
    findMissing(a, b, n, m);
    return 0;
}
```

Output :

```
5 6
```

Time complexity : O(n)
Auxiliary Space : O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-elements-present-first-array-not-second/

Chapter 93

Find four elements a, b, c and d in an array such that a+b = c+d

Find four elements a, b, c and d in an array such that a+b = c+d - GeeksforGeeks

Given an array of distinct integers, find if there are two pairs (a, b) and (c, d) such that a+b = c+d, and a, b, c and d are distinct elements. If there are multiple answers, then print any of them.

Example:

```
Input:   {3, 4, 7, 1, 2, 9, 8}
Output:  (3, 8) and (4, 7)
Explanation: 3+8 = 4+7

Input:   {3, 4, 7, 1, 12, 9};
Output:  (4, 12) and (7, 9)
Explanation: 4+12 = 7+9

Input:  {65, 30, 7, 90, 1, 9, 8};
Output:  No pairs found
```

Expected Time Complexity: $O(n^2)$

A **Simple Solution** is to run four loops to generate all possible quadruples of array element. For every quadruple (a, b, c, d), check if (a+b) = (c+d). Time complexity of this solution is $O(n^4)$.

An **Efficient Solution** can solve this problem in $O(n^2)$ time. The idea is to use hashing. We use sum as key and pair as value in hash table.

```
Loop i = 0 to n-1 :
    Loop j = i + 1 to n-1 :
        calculate sum
        If in hash table any index already exist
            Then print (i, j) and previous pair
            from hash table
        Else update hash table
    EndLoop;
EndLoop;
```

Below are implementations of above idea. In below implementation, map is used instead of hash. Time complexity of map insert and search is actually O(Log n) instead of O(1). So below implementation is O(n^2 Log n).

C/C++

```
 // Find four different elements a,b,c and d of array such that
// a+b = c+d
#include<bits/stdc++.h>
using namespace std;

bool findPairs(int arr[], int n)
{
    // Create an empty Hash to store mapping from sum to
    // pair indexes
    map<int, pair<int, int> > Hash;

    // Traverse through all possible pairs of arr[]
    for (int i = 0; i < n; ++i)
    {
        for (int j = i + 1; j < n; ++j)
        {
            // If sum of current pair is not in hash,
            // then store it and continue to next pair
            int sum = arr[i] + arr[j];
            if (Hash.find(sum) == Hash.end())
                Hash[sum] = make_pair(i, j);

            else // Else (Sum already present in hash)
            {
                // Find previous pair
                pair<int, int> pp = Hash[sum];// pp->previous pair

                // Since array elements are distinct, we don't
                // need to check if any element is common among pairs
                cout << "(" << arr[pp.first] << ", " << arr[pp.second]
                     << ") and (" << arr[i] << ", " << arr[j] << ")n";
```

```
                return true;
            }
        }
    }

    cout << "No pairs found";
    return false;
}

// Driver program
int main()
{
    int arr[] = {3, 4, 7, 1, 2, 9, 8};
    int n  =  sizeof arr / sizeof arr[0];
    findPairs(arr, n);
    return 0;
}
```

Java

```
 // Java Program to find four different elements a,b,c and d of
// array such that a+b = c+d
import java.io.*;
import java.util.*;

class ArrayElements
{
    // Class to represent a pair
    class pair
    {
        int first, second;
        pair(int f,int s)
        {
            first = f; second = s;
        }
    };

    boolean findPairs(int arr[])
    {
        // Create an empty Hash to store mapping from sum to
        // pair indexes
        HashMap<Integer,pair> map = new HashMap<Integer,pair>();
        int n=arr.length;

        // Traverse through all possible pairs of arr[]
        for (int i=0; i<n; ++i)
        {
            for (int j=i+1; j<n; ++j)
```

```
            {
                // If sum of current pair is not in hash,
                // then store it and continue to next pair
                int sum = arr[i]+arr[j];
                if (!map.containsKey(sum))
                    map.put(sum,new pair(i,j));

                else // Else (Sum already present in hash)
                {
                    // Find previous pair
                    pair p = map.get(sum);

                    // Since array elements are distinct, we don't
                    // need to check if any element is common among pairs
                    System.out.println("("+arr[p.first]+", "+arr[p.second]+
                                       ") and ("+arr[i]+", "+arr[j]+")");
                    return true;
                }
            }
        }
        return false;
    }

    // Testing program
    public static void main(String args[])
    {
        int arr[] = {3, 4, 7, 1, 2, 9, 8};
        ArrayElements a = new ArrayElements();
        a.findPairs(arr);
    }
}
// This code is contributed by Aakash Hasija
```

Python

```
 # Java Program to find four different elements a,b,c and d of
# array such that a+b = c+d

# function to find a, b, c, d such that
# (a + b) = (c + d)
def findPairs(arr, n):

# Create an empty hashmap to store mapping
# from sum to pair indexes
Hash = {}

# Traverse through all possible pairs of arr[]
for i in range(n - 1):
```

```
    for j in range(i + 1, n):
    sum = arr[i] + arr[j]
    # Sum already present in hash
    if sum in Hash.keys():
        # print previous pair and current
        prev = Hash.get(sum)
        print (str(prev) + " and (%d, %d)"
        %(arr[i], arr[j]))
        return True
    else:
        # sum is not in hash
        # store it and continue to next pair
        Hash[sum] = (arr[i], arr[j])
return False

# driver program
arr = [3, 4, 7, 1, 2, 9, 8]
n = len(arr)
findPairs(arr, n)

# This code is contributed by Aditi Sharma
```

Output:

```
(3, 8) and (4, 7)
```

Thanks to Gaurav Ahirwar for suggesting above solutions.

Exercise:
1) Extend the above solution with duplicates allowed in array.
2) Further extend the solution to print all quadruples in output instead of just one. And all quadruples should be printed printed in lexicographical order (smaller values before greater ones). Assume we have two solutions S1 and S2.

```
S1 : a1 b1 c1 d1 ( these are values of indices int the array )
S2 : a2 b2 c2 d2

S1 is lexicographically smaller than S2 iff
  a1 < a2 OR
  a1 = a2 AND b1 < b2 OR
  a1 = a2 AND b1 = b2 AND c1 < c2 OR
  a1 = a2 AND b1 = b2 AND c1 = c2 AND d1 < d2
```

See this for solution of exercise.

Related Article :
Find all pairs (a,b) and (c,d) in array which satisfy ab = cd

Source

https://www.geeksforgeeks.org/find-four-elements-a-b-c-and-d-in-an-array-such-that-ab-cd/

Chapter 94

Find four elements that sum to a given value | Set 2 (O(n^2Logn) Solution)

Find four elements that sum to a given value | Set 2 (O(n^2Logn) Solution) - GeeksforGeeks

Given an array of integers, find any one combination of four elements in the array whose sum is equal to a given value X.
For example, if the given array is {10, 2, 3, 4, 5, 9, 7, 8} and X = 23, then your function should print "3 5 7 8" (3 + 5 + 7 + 8 = 23).

We have discussed a O(n^3) algorithm in the previous post on this topic. The problem can be solved in O(n^2Logn) time with the help of auxiliary space.

Thanks to itsnimish for suggesting this method. Following is the detailed process.

Let the input array be A[].

1) Create an auxiliary array aux[] and store sum of all possible pairs in aux[]. The size of aux[] will be n*(n-1)/2 where n is the size of A[].

2) Sort the auxiliary array aux[].

3) Now the problem reduces to find two elements in aux[] with sum equal to X. We can use method 1 ofthis postto find the two elements efficiently. There is following important point to note though. An element of aux[] represents a pair from A[]. While picking two elements from aux[], we must check whether the two elements have an element of A[] in common. For example, if first element sum of A[1] and A[2], and second element is sum of A[2] and A[4], then these two elements of aux[] don't represent four distinct elements of input array A[].

Following is C implementation of this method.

```
 #include <stdio.h>
#include <stdlib.h>
```

```
// The following structure is needed to store pair sums in aux[]
struct pairSum
{
    int first; // index (int A[]) of first element in pair
    int sec; // index of second element in pair
    int sum;  // sum of the pair
};

// Following function is needed for library function qsort()
int compare (const void *a, const void * b)
{
    return ( (*(pairSum *)a).sum - (*(pairSum*)b).sum );
}

// Function to check if two given pairs have any common element or not
bool noCommon(struct pairSum a, struct pairSum b)
{
    if (a.first == b.first || a.first == b.sec ||
            a.sec == b.first || a.sec == b.sec)
        return false;
    return true;
}


// The function finds four elements with given sum X
void findFourElements (int arr[], int n, int X)
{
    int i, j;

    // Create an auxiliary array to store all pair sums
    int size = (n*(n-1))/2;
    struct pairSum aux[size];

    /* Generate all possible pairs from A[] and store sums
       of all possible pairs in aux[] */
    int k = 0;
    for (i = 0; i < n-1; i++)
    {
        for (j = i+1; j < n; j++)
        {
            aux[k].sum = arr[i] + arr[j];
            aux[k].first = i;
            aux[k].sec = j;
            k++;
        }
    }
```

```
    // Sort the aux[] array using library function for sorting
    qsort (aux, size, sizeof(aux[0]), compare);

    // Now start two index variables from two corners of array
    // and move them toward each other.
    i = 0;
    j = size-1;
    while (i < size && j >=0 )
    {
        if ((aux[i].sum + aux[j].sum == X) && noCommon(aux[i], aux[j]))
        {
            printf ("%d, %d, %d, %d\n", arr[aux[i].first], arr[aux[i].sec],
                                        arr[aux[j].first], arr[aux[j].sec]);
            return;
        }
        else if (aux[i].sum + aux[j].sum < X)
            i++;
        else
            j--;
    }
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 20, 30, 40, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    int X = 91;
    findFourElements (arr, n, X);
    return 0;
}
```

Output:

```
20, 1, 30, 40
```

Please note that the above code prints only one quadruple. If we remove the return statement and add statements "i++; j–;", then it prints same quadruple five times. The code can modified to print all quadruples only once. It has been kept this way to keep it simple.

Time complexity: The step 1 takes O(n^2) time. The second step is sorting an array of size O(n^2). Sorting can be done in O(n^2Logn) time using merge sort or heap sort or any other O(nLogn) algorithm. The third step takes O(n^2) time. So overall complexity is O(n^2Logn).

Auxiliary Space: O(n^2). The big size of auxiliary array can be a concern in this method.

1. Store sums of all pairs in a hash table

2. Traverse through all pairs again and search for X – (current pair sum) in the hash table.
3. If a pair is found with the required sum, then make sure that all elements are distinct array elements and an element is not considered more than once.

```
 // A hashing based  CPP program to find if there are
// four elements with given sum.
#include <bits/stdc++.h>
using namespace std;

// The function finds four elements with given sum X
void findFourElements (int arr[], int n, int X)
{
    // Store sums of all pairs in a hash table
    unordered_map<int, pair<int, int>> mp;
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            mp[arr[i] + arr[j]] = {i, j};

    // Traverse through all pairs and search
    // for X - (current pair sum).
    for (int i = 0; i < n-1; i++)
    {
        for (int j = i+1; j < n; j++)
        {
            int sum = arr[i] + arr[j];

            // If X - sum is present in hash table,
            if (mp.find(X - sum) != mp.end())
            {

                // Making sure that all elements are
                // distinct array elements and an element
                // is not considered more than once.
                pair<int, int> p = mp[X - sum];
                if (p.first != i && p.first != j &&
                        p.second != i && p.second != j)
                {
                    cout << arr[i] << ", " << arr[j] << ", "
                         << arr[p.first] << ", "
                         << arr[p.second];
                    return;
                }
            }
        }
    }
}
```

```
// Driver program to test above function
int main()
{
    int arr[] = {10, 20, 30, 40, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    int X = 91;
    findFourElements(arr, n, X);
    return 0;
}
```

Improved By : Mrinal Tak, ImStark

Source

https://www.geeksforgeeks.org/find-four-elements-that-sum-to-a-given-value-set-2/

Chapter 95

Find four elements that sum to a given value | Set 3 (Hashmap)

Find four elements that sum to a given value | Set 3 (Hashmap) - GeeksforGeeks

Given an array of integers, Check if there exist four elements at different indexes in the array whose sum is equal to a given value k.
For example, if the given array is {1 5 1 0 6 0} and k = 7, then your function should print "YES" as (1+5+1+0=7).

Examples:

```
Input  : arr[] = {1 5 1 0 6 0}
             k = 7
Output : YES

Input :  arr[] = {38 7 44 42 28 16 10 37
                  33 2 38 29 26 8 25}
           k = 22
Output : NO
```

We have discussed different solutions in below two sets.
Find four elements that sum to a given value | Set 1 (n^3 solution)
Find four elements that sum to a given value | Set 2 (O(n^2Logn) Solution)

In this post, an optimized solution is discussed that works in $O(n^2)$ on average.

The idea is to create a hashmap to store pair sums.

```
Loop i = 0 to n-1 :
 Loop j = i + 1 to n-1
```

```
   calculate sum = arr[i] + arr[j]
     If (k-sum) exist in hash
      a) Check in hash table for all
         pairs of indexes which form
         (k-sum).
      b) If there is any pair with no
         no common indexes.
           return true
    Else  update hash table
    EndLoop;
EndLoop;
```

```
 // C++ program to find if there exist 4 elements
// with given sum
#include <bits/stdc++.h>
using namespace std;

// function to check if there exist four
// elements whose sum is equal to k
bool findfour(int arr[], int n, int k)
{
    // map to store sum and indexes for
    // a pair sum
    unordered_map<int, vector<pair<int, int> > > hash;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {

            // calculate the sum of each pair
            int sum = arr[i] + arr[j];

            // if k-sum exist in map
            if (hash.find(k - sum) != hash.end()) {
                auto num = hash.find(k - sum);
                vector<pair<int, int> > v = num->second;

                // check for index coincidence as if
                // there is a common that means all
                // the four numbers are not from
                // different indexes and one of the
                // index is repeated
                for (int k = 0; k < num->second.size(); k++) {

                    pair<int, int> it = v[k];

                    // if all indexes are different then
                    // it means four number exist
                    // set the flag and break the loop
```

```
                    if (it.first != i && it.first != j &&
                        it.second != i && it.second != j)
                        return true;
                }
            }

            // store the sum and index pair in hashmap
            hash[sum].push_back(make_pair(i, j));
        }
    }
    hash.clear();
    return false;
}

// Driver code
int main()
{
    int k = 7;
    int arr[] = { 1, 5, 1, 0, 6, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (findfour(arr, n, k))
        cout
            << "YES" << endl;
    else
        cout << "NO" << endl;
    return 0;
}
```

Output:

```
YES
```

Source

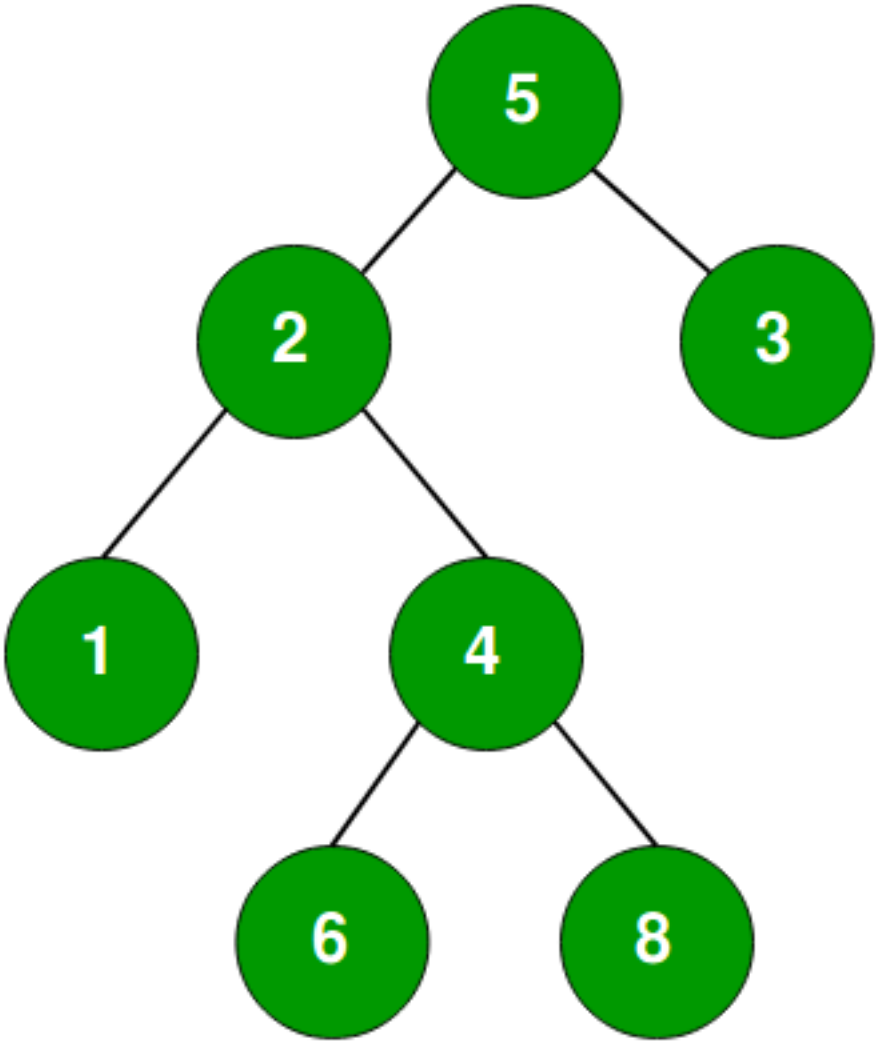
https://www.geeksforgeeks.org/find-four-elements-sum-given-value-set-3-hashmap/

Chapter 96

Find if there is a pair in root to a leaf path with sum equals to root's data

Find if there is a pair in root to a leaf path with sum equals to root's data - GeeksforGeeks

Given a binary tree, find if there is a pair in root to a leaf path such that sum of values in pair is equal to root's data. For example, in below tree (2, 3) and (4, 1) are pairs with sum equals to root's data.



The idea is based on hashing and tree traversal. The idea is similar to method 2 of array pair sum problem.

- Create an empty hash table.
- Start traversing tree in Preorder fashion.
- If we reach a leaf node, we return false.
- For every visited node, check if root's data minus current node's data exists in hash table or not. If yes, return true. Else insert current node in hash table.
- Recursively check in left and right subtrees.
- Remove current node from hash table so that it doesn't appear in other root to leaf paths.

Below is C++ implementation of above idea.

```
 // C++ program to find if there is a pair in any root
// to leaf path with sum equals to root's key.
#include<bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node
{
    int data;
    struct Node* left, *right;
};

/* utility that allocates a new node with the
given data and NULL left and right pointers. */
struct Node* newnode(int data)
{
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right  = NULL;
    return (node);
}

// Function to print root to leaf path which satisfies the condition
bool printPathUtil(Node *node, unordered_set<int> &s, int root_data)
{
    // Base condition
    if (node == NULL)
        return false;

    // Check if current node makes a pair with any of the
    // existing elements in set.
    int rem = root_data - node->data;
    if (s.find(rem) != s.end())
        return true;

    // Insert current node in set
    s.insert(node->data);

    // If result returned by either left or right child is
    // true, return true.
    bool res = printPathUtil(node->left, s, root_data) ||
               printPathUtil(node->right, s, root_data);

    // Remove current node from hash table
    s.erase(node->data);
```

```
    return res;
}

// A wrapper over printPathUtil()
bool isPathSum(Node *root)
{
   // create an empty hash table
   unordered_set<int> s;

   // Recursively check in left and right subtrees.
   return printPathUtil(root->left, s, root->data) ||
          printPathUtil(root->right, s, root->data);
}

// Driver program to run the case
int main()
{
    struct Node *root = newnode(8);
    root->left    = newnode(5);
    root->right   = newnode(4);
    root->left->left = newnode(9);
    root->left->right = newnode(7);
    root->left->right->left = newnode(1);
    root->left->right->right = newnode(12);
    root->left->right->right->right = newnode(2);
    root->right->right = newnode(11);
    root->right->right->left = newnode(3);
    isPathSum(root)? cout << "Yes" : cout << "No";
    return 0;
}
```

Output:

```
Yes
```

Time Complexity : O(n) under the assumption that hash search, insert and erase take O(1) time.

Exercise : Extend the above solution to print all root to leaf paths that have a pair with sum equals to root's data.

Source

https://www.geeksforgeeks.org/find-pair-root-leaf-path-sum-equals-roots-data/

Chapter 97

Find if there is a rectangle in binary matrix with corners as 1

Find if there is a rectangle in binary matrix with corners as 1 - GeeksforGeeks

There is a given binary matrix, we need to find if there exists any rectangle or square in the given matrix whose all four corners are equal to 1.

Examples:

```
Input :
mat[][] = { 1 0 0 1 0
            0 0 1 0 1
            0 0 0 1 0
            1 0 1 0 1}
Output : Yes
as there exists-
1 0 1
0 1 0
1 0 1
```

Brute Force Approach-
We start scanning the matrix whenever we find a 1 at any index then we try for all the combination for index with which we can form the rectangle.
algorithm-

```
for i = 1 to rows
  for j = 1 to columns
   if matrix[i][j] == 1
     for k=i+1 to rows
```

```
       for l=j+1 to columns
         if (matrix[i][k]==1 &&
             matrix[l][i]==1 &&
             m[l][k]==1)
                return true
return false
```

-Time Complexity of this solutions- O(m^2*n^2)

```
 // A brute force approach based CPP program to
// find if there is a rectangle with 1 as corners.
#include <bits/stdc++.h>
using namespace std;

// Returns true if there is a rectangle with
// 1 as corners.
bool isRectangle(const vector<vector<int> >& m)
{
    // finding row and column size
    int rows = m.size();
    if (rows == 0)
        return false;
    int columns = m[0].size();

    // scanning the matrix
    for (int y1 = 0; y1 < rows; y1++)
      for (int x1 = 0; x1 < columns; x1++)

        // if any index found 1 then try
        // for all rectangles
        if (m[y1][x1] == 1)
          for (int y2 = y1 + 1; y2 < rows; y2++)
            for (int x2 = x1 + 1; x2 < columns; x2++)
               if (m[y1][x2] == 1 &&
                   m[y2][x1] == 1 &&
                     m[y2][x2] == 1)
                             return true;
    return false;
}

// Driver code
int main()
{
    vector<vector<int> > mat = { { 1, 0, 0, 1, 0 },
                                 { 0, 0, 1, 0, 1 },
                                 { 0, 0, 0, 1, 0 },
                                 { 1, 0, 1, 0, 1 } };
    if (isRectangle(mat))
```

```
        cout << "Yes";
    else
        cout << "No";
}
```

Output:

```
Yes
```

Efficient Approach
– Scan from top to down, line by line
– For each line, remember each combination of 2 1's and push that into a hash-set
– If we ever find that combination again in a later line, we get our rectangle
– Time Complexity of this solution- O(n*m^2)

```
 // An efficient approach based CPP program to
// find if there is a rectangle with 1 as
// corners.
#include <bits/stdc++.h>
using namespace std;

// Returns true if there is a rectangle with
// 1 as corners.
bool isRectangle(const vector<vector<int> >& matrix)
{
    // finding row and column size
    int rows = matrix.size();
    if (rows == 0)
        return false;

    int columns = matrix[0].size();

    // map for storing the index of combination of 2 1's
    unordered_map<int, unordered_set<int> > table;

    // scanning from top to bottom line by line
    for (int i = 0; i < rows; ++i) {

     for (int j = 0; j < columns - 1; ++j) {
        for (int k = j + 1; k < columns; ++k) {

          // if found two 1's in a column
          if (matrix[i][j] == 1 &&
              matrix[i][k] == 1) {

            // check if there exists 1's in same
```

```
            // row previously then return true
            if (table.find(j) != table.end() &&
                table[j].find(k) != table[j].end())
                        return true;

            if (table.find(k) != table.end() &&
                table[k].find(j) != table[k].end())
                        return true;

            // store the indexes in hashset
            table[j].insert(k);
            table[k].insert(j);
          }
        }
      }
    }
    return false;
}

// Driver code
int main()
{
    vector<vector<int> > mat = { { 1, 0, 0, 1, 0 },
                                 { 0, 0, 1, 0, 1 },
                                 { 0, 0, 0, 1, 0 },
                                { 1, 0, 1, 0, 1 } };
    if (isRectangle(mat))
        cout << "Yes";
    else
        cout << "No";
}
```

Output:

```
Yes
```

Improved By : abhishekahuja02

Source

https://www.geeksforgeeks.org/find-rectangle-binary-matrix-corners-1/

Chapter 98

Find if there is a subarray with 0 sum

Find if there is a subarray with 0 sum - GeeksforGeeks

Given an array of positive and negative numbers, find if there is a subarray (of size at-least one) with 0 sum.

Examples :

```
Input: {4, 2, -3, 1, 6}
Output: true
There is a subarray with zero sum from index 1 to 3.

Input: {4, 2, 0, 1, 6}
Output: true
There is a subarray with zero sum from index 2 to 2.

Input: {-3, 2, 3, 1, 6}
Output: false
There is no subarray with zero sum.
```

A **simple solution** is to consider all subarrays one by one and check the sum of every subarray. We can run two loops: the outer loop picks a starting point i and the inner loop tries all subarrays starting from i (See thisfor implementation). Time complexity of this method is $O(n^2)$.

We can also **use hashing**. The idea is to iterate through the array and for every element arr[i], calculate sum of elements form 0 to i (this can simply be done as sum += arr[i]). If the current sum has been seen before, then there is a zero sum array. Hashing is used to store the sum values, so that we can quickly store sum and find out whether the current sum is seen before or not.

Example :

```
arr[] = {1, 4, -2, -2, 5, -4, 3}

If we consider all prefix sums, we can
notice that there is a subarray with 0
sum when :
1) Either a prefix sum repeats or
2) Or prefix sum becomes 0.

Prefix sums for above array are:
1, 5, 3, 1, 6, 2, 5

Since prefix sum 1 repeats, we have a subarray
with 0 sum.
```

Following is implementation of the above approach.

C++

```
 // A C++ program to find if there is a zero sum
// subarray
#include <bits/stdc++.h>
using namespace std;

bool subArrayExists(int arr[], int n)
{
    unordered_set<int> sumSet;

    // Traverse through array and store prefix sums
    int sum = 0;
    for (int i = 0 ; i < n ; i++)
    {
        sum += arr[i];

        // If prefix sum is 0 or it is already present
        if (sum == 0 || sumSet.find(sum) != sumSet.end())
            return true;

        sumSet.insert(sum);
    }
    return false;
}

// Driver code
int main()
{
    int arr[] =  {-3, 2, 3, 1, 6};
```

```
    int n = sizeof(arr)/sizeof(arr[0]);
    if (subArrayExists(arr, n))
        cout << "Found a subarray with 0 sum";
    else
        cout << "No Such Sub Array Exists!";
    return 0;
}
```

Java

```
 // A Java program to find if there is a zero sum subarray
import java.util.HashMap;

class ZeroSumSubarray {

    // Returns true if arr[] has a subarray with sero sum
    static Boolean subArrayExists(int arr[])
    {
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM =
                        new HashMap<Integer, Integer>();

        // Initialize sum of elements
        int sum = 0;

        // Traverse through the given array
        for (int i = 0; i < arr.length; i++)
        {
            // Add current element to sum
            sum += arr[i];

            // Return true in following cases
            // a) Current element is 0
            // b) sum of elements from 0 to i is 0
            // c) sum is already present in hash map
            if (arr[i] == 0 || sum == 0 || hM.get(sum) != null)
                return true;

            // Add sum to hash map
            hM.put(sum, i);
        }

        // We reach here only when there is
        // no subarray with 0 sum
        return false;
    }

    // driver code
```

```
    public static void main(String arg[])
    {
        int arr[] = {-3, 2, 3, 1, 6};
        if (subArrayExists(arr))
            System.out.println("Found a subarray with 0 sum");
        else
            System.out.println("No Such Sub Array Exists!");
    }
}
```

C#

```
 // A C# program to find if there
// is a zero sum subarray
using System;
using System.Collections.Generic;

class GFG
{
// Returns true if arr[] has
// a subarray with sero sum
    static Boolean subArrayExists(int []arr)
    {
        // Creates an empty hashMap hM
        Dictionary<int,
                   int> hM = new Dictionary<int,
                                            int>();
        // Initialize sum of elements
        int sum = 0;

        // Traverse through the given array
        for (int i = 0; i < arr.Length; i++)
        {
            // Add current element to sum
            sum += arr[i];

            // Return true in following cases
            // a) Current element is 0
            // b) sum of elements from 0 to i is 0
            // c) sum is already present in hash map
            if (arr[i] == 0 || sum == 0 )
                return true;

            // Add sum to hash map
            hM[i] = sum;
        }

        // We reach here only when there is
```

```
        // no subarray with 0 sum
        return false;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {-3, 2, 3, 1, 6};
        if (subArrayExists(arr))
            Console.WriteLine("Found a subarray " +
                                    "with 0 sum");
        else
            Console.WriteLine("No Such Sub " +
                            "Array Exists!");
    }
}

// This code is contributed by Sam007
```

Output :

```
No Such Sub Array Exists!
```

Time Complexity of this solution can be considered as O(n) under the assumption that we have good hashing function that allows insertion and retrieval operations in O(1) time.

Exercise:
Extend the above program to print starting and ending indexes of all subarrays with 0 sum.

This article is contributed by **Chirag Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007

Source

https://www.geeksforgeeks.org/find-if-there-is-a-subarray-with-0-sum/

Chapter 99

Find k numbers with most occurrences in the given array

Find k numbers with most occurrences in the given array - GeeksforGeeks

Given an array of **n** numbers and a positive integer **k**. The problem is to find **k** numbers with most occurrences, i.e., the top **k** numbers having the maximum frequency. If two numbers have same frequency then the larger number should be given preference. The numbers should be displayed in decreasing order of their frequencies. It is assumed that the array consists of **k** numbers with most occurrences.

Examples:

```
Input : arr[] = {3, 1, 4, 4, 5, 2, 6, 1},
             k = 2
Output : 4 1
Frequency of 4 = 2
Frequency of 1 = 2
These two have the maximum frequency and
4 is larger than 1.

Input : arr[] = {7, 10, 11, 5, 2, 5, 5, 7, 11, 8, 9},
            k = 4
Output : 5 11 7 10
```

Asked in Amazon Interview

Method 1: Using hash table, we create a frequency table which stores the frequency of occurrence of each number in the given array. In the hash table we define **(x, y)** tuple, where **x** is the key(number) and **y** is its frequency in the array. Now we traverse this hash table and create an array **freq_arr**[] which stores these (number, frequency) tuples. Sort this **freq_arr**[] on the basis of the conditions defined in the problem statement. Now, print the first **k** numbers of this **freq_arr**[].

```
 // C++ implementation to find k numbers with most
// occurrences in the given array
#include <bits/stdc++.h>

using namespace std;

// comparison function to sort the 'freq_arr[]'
bool compare(pair<int, int> p1, pair<int, int> p2)
{
    // if frequencies of two elements are same
    // then the larger number should come first
    if (p1.second == p2.second)
        return p1.first > p2.first;

    // sort on the basis of decreasing order
    // of frequencies
    return p1.second > p2.second;
}

// funnction to print the k numbers with most occurrences
void print_N_mostFrequentNumber(int arr[], int n, int k)
{
    // unordered_map 'um' implemented as frequency hash table
    unordered_map<int, int> um;
    for (int i = 0; i<n; i++)
        um[arr[i]]++;

    // store the elements of 'um' in the vector 'freq_arr'
    vector<pair<int, int> > freq_arr(um.begin(), um.end());

    // sort the vector 'freq_arr' on the basis of the
    // 'compare' function
    sort(freq_arr.begin(), freq_arr.end(), compare);

    // display the the top k numbers
    cout << k << " numbers with most occurrences are:\n";
    for (int i = 0; i<k; i++)
        cout << freq_arr[i].first << " ";
}

// Driver program to test above
int main()
{
    int arr[] = {3, 1, 4, 4, 5, 2, 6, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    print_N_mostFrequentNumber(arr, n, k);
    return 0;
```

```
}
```

Output:

```
2 numbers with most occurrences are:
4 1
```

Time Complexity: O(dlogd), where **d** is the count of distinct elements in the array.
Auxiliary Space: O(d), where **d** is the count of distinct elements in the array.

Method 2: Create the array **freq_arr**[] as described in **Method 1** of this post. Now, build the max heap using elements of this **freq_arr**[]. The root of the max heap should be the most frequent number and in case of conflicts the larger number gets the preference. Now remove the top **k** numbers of this max heap. C++ STL priority_queue has been used as max heap.

```
 // C++ implementation to find k numbers with most
// occurrences in the given array
#include <bits/stdc++.h>

using namespace std;

// comparison function defined for the priority queue
struct compare
{
    bool operator()(pair<int, int> p1, pair<int, int> p2)
    {
        // if frequencies of two elements are same
        // then the larger number should come first
        if (p1.second == p2.second)
            return p1.first < p2.first;

        // insert elements in the priority queue on the basis of
        // decreasing order of frequencies
        return p1.second < p2.second;
    }
};

// funnction to print the k numbers with most occurrences
void print_N_mostFrequentNumber(int arr[], int n, int k)
{
    // unordered_map 'um' implemented as frequency hash table
    unordered_map<int, int> um;
    for (int i = 0; i<n; i++)
        um[arr[i]]++;

    // store the elements of 'um' in the vector 'freq_arr'
```

```
    vector<pair<int, int> > freq_arr(um.begin(), um.end());

    // priority queue 'pq' implemented as max heap on the basis
    // of the comparison operator 'compare'
    // element with the highest frequency is the root of 'pq'
    // in case of conflicts, larger element is the root
    priority_queue<pair<int, int>, vector<pair<int, int> >,
                           compare> pq(um.begin(), um.end());

    // display the the top k numbers
    cout << k << " numbers with most occurrences are:\n";
    for (int i = 1; i<= k; i++)
    {
        cout << pq.top().first << " ";
        pq.pop();
    }
}

// Driver program to test above
int main()
{
    int arr[] = {3, 1, 4, 4, 5, 2, 6, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    print_N_mostFrequentNumber(arr, n, k);
    return 0;
}
```

Output:

```
2 numbers with most occurrences are:
4 1
```

Time Complexity: O(klogd), where **d** is the count of distinct elements in the array.
Auxiliary Space: O(d), where **d** is the count of distinct elements in the array.

References: https://www.careercup.com/question?id=5082885552865280

Source

https://www.geeksforgeeks.org/find-k-numbers-occurrences-given-array/

Chapter 100

Find largest d in array such that a + b + c = d

Find largest d in array such that a + b + c = d - GeeksforGeeks

Given a set S (all distinct elements) of integers, find the largest d such that a + b + c = d where a, b, c, and d are distinct elements of S.

```
Constraints:
1   number of elements in the set   1000
INT_MIN   each element in the set   INT_MAX
```

Examples :

Input : S[] = {2, 3, 5, 7, 12}
Output : 12
Explanation: 12 is the largest d which can be represented as 12 = 2 + 3 + 7

Input : S[] = {2, 16, 64, 256, 1024}
Output : No solution

Method 1(Brute Force)
We can solve this problem using simple brute force approach which is not very efficient as |S| can be as large as 1000. We'll sort the set of elements and start by finding the largest d by equating it with the sum of all possible combinations of a, b and c.
Below is the implementation of above idea :

C++

```
 // CPP Program to find the largest d
// such that d = a + b + c
#include <bits/stdc++.h>
```

```
using namespace std;

int findLargestd(int S[], int n)
{
    bool found = false;

    // sort the array in
    // ascending order
    sort(S, S + n);

    // iterating from backwards to
    // find the required largest d
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = 0; j < n; j++)
        {

            // since all four a, b, c,
            // d should be distinct
            if (i == j)
                continue;

            for (int k = j + 1; k < n; k++)
            {
                if (i == k)
                    continue;

                for (int l = k + 1; l < n; l++)
                {
                    if (i == l)
                        continue;

                    // if the current combination
                    // of j, k, l in the set is
                    // equal to S[i] return this
                    // value as this would be the
                    // largest d since we are
                    //  iterating in descending order
                    if (S[i] == S[j] + S[k] + S[l])
                    {
                        found = true;
                        return S[i];
                    }
                }
            }
        }
    }
    if (found == false)
```

```
        return INT_MIN;
}

// Driver Code
int main()
{
    // Set of distinct Integers
    int S[] = { 2, 3, 5, 7, 12 };
    int n = sizeof(S) / sizeof(S[0]);

    int ans = findLargestd(S, n);
    if (ans == INT_MIN)
        cout << "No Solution" << endl;
    else
        cout << "Largest d such that a + b + "
             << "c = d is " << ans << endl;
    return 0;
}
```

Java

```
 // Java Program to find the largest
// such that d = a + b + c
import java.io.*;
import java.util.Arrays;

class GFG
{

// function to find largest d
static int findLargestd(int []S, int n)
{
    boolean found = false;

    // sort the array in
    // ascending order
    Arrays.sort(S);

    // iterating from backwards to
    // find the required largest d
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = 0; j < n; j++)
        {

            // since all four a, b, c,
            // d should be distinct
            if (i == j)
```

```
                continue;

            for (int k = j + 1; k < n; k++)
            {
                if (i == k)
                    continue;

                for (int l = k + 1; l < n; l++)
                {
                    if (i == l)
                        continue;

                    // if the current combination
                    // of j, k, l in the set is
                    // equal to S[i] return this
                    // value as this would be the
                    // largest d since we are
                    // iterating in descending order
                    if (S[i] == S[j] + S[k] + S[l])
                    {
                        found = true;
                        return S[i];
                    }
                }
            }
        }
    }
    if (found == false)
        return Integer.MAX_VALUE;

    return -1;
}

// Driver Code
public static void main(String []args)
{
    // Set of distinct Integers
    int []S = new int[]{ 2, 3, 5, 7, 12 };
    int n = S.length;

    int ans = findLargestd(S, n);
    if (ans == Integer.MAX_VALUE)
        System.out.println("No Solution");
    else
        System.out.println("Largest d such that " +
                        "a + " + "b + c = d is " +
                                            ans );
```

```
}
}

// This code is contributed by Sam007
```

Python3

```
 # Python Program to find the largest
# d such that d = a + b + c

def findLargestd(S, n) :
    found = False

    # sort the array in ascending order
    S.sort()

    # iterating from backwards to
    # find the required largest d
    for i in range(n-1, -1, -1) :
        for j in range(0, n) :

            # since all four a, b, c,
            # d should be distinct
            if (i == j) :
                continue

            for k in range(j + 1, n) :
                if (i == k) :
                    continue

                for l in range(k+1, n) :
                    if (i == l) :
                        continue

                    # if the current combination
                    # of j, k, l in the set is
                    # equal to S[i] return this
                    # value as this would be the
                    # largest d since we are
                    # iterating in descending order
                    if (S[i] == S[j] + S[k] + S[l]) :
                        found = True
                        return S[i]

    if (found == False) :
        return -1

# Driver Code
```

```
# Set of distinct Integers
S = [ 2, 3, 5, 7, 12 ]
n = len(S)

ans = findLargestd(S, n)
if (ans == -1) :
    print ("No Solution")
else :
    print ("Largest d such that a + b +" ,
        "c = d is" ,ans)

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
 // C# Program to find the largest
// such that d = a + b + c
using System;

class GFG
{

// function to find largest d
static int findLargestd(int []S,
                        int n)
{
    bool found = false;

    // sort the array
    // in ascending order
    Array.Sort(S);

    // iterating from backwards to
    // find the required largest d
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = 0; j < n; j++)
        {

            // since all four a, b, c,
            // d should be distinct
            if (i == j)
                continue;

            for (int k = j + 1; k < n; k++)
            {
```

```
                if (i == k)
                    continue;

                for (int l = k + 1; l < n; l++)
                {
                    if (i == l)
                        continue;

                    // if the current combination
                    // of j, k, l in the set is
                    // equal to S[i] return this
                    // value as this would be the
                    // largest dsince we are
                    // iterating in descending order
                    if (S[i] == S[j] + S[k] + S[l])
                    {
                        found = true;
                        return S[i];
                    }
                }
            }
        }
    }
    if (found == false)
        return int.MaxValue;

    return -1;
}

// Driver Code
public static void Main()
{
    // Set of distinct Integers
    int []S = new int[]{ 2, 3, 5, 7, 12 };
    int n = S.Length;

    int ans = findLargestd(S, n);
    if (ans == int.MaxValue)
        Console.WriteLine( "No Solution");
    else
        Console.Write("Largest d such that a + " +
                          "b + c = d is " + ans );

}
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP Program to find the largest
// d such that d = a + b + c

function findLargestd( $S, $n)
{
$found = false;

    // sort the array in
    // ascending order
    sort($S);

    // iterating from backwards to
    // find the required largest d
    for ( $i = $n - 1; $i >= 0; $i--)
    {
        for ( $j = 0; $j < $n; $j++)
        {

            // since all four a, b, c,
            // d should be distinct
            if ($i == $j)
                continue;

            for ( $k = $j + 1; $k < $n; $k++)
            {
                if ($i == $k)
                    continue;

                for ( $l = $k + 1; $l < $n; $l++)
                {
                    if ($i == $l)
                        continue;

                    // if the current combination
                    // of j, k, l in the set is
                    // equal to S[i] return this
                    // value as this would be the
                    // largest d since we are
                    // iterating in descending order
                    if ($S[$i] == $S[$j] + $S[$k] + $S[$l])
                    {
                        $found = true;
                        return $S[$i];
                    }
                }
```

```
            }
        }
    }
    if ($found == false)
        return PHP_INT_MIN;
}

// Driver Code

// Set of distinct Integers
$S = array( 2, 3, 5, 7, 12 );
$n = count($S);

$ans = findLargestd($S, $n);
if ($ans == PHP_INT_MIN)
    echo "No Solution" ;
else
    echo "Largest d such that a + b + " ,
         "c = d is " , $ans ;

// This code is contributed by anuj_67.
?>
```

Output :

```
Largest d such that a + b + c = d is 12
```

This brute force solution has a time complexity of O((size of Set)^4).

Method 2(Efficient Approach – Using Hashing)
The above problem statement (a + b + c = d) can be restated as finding a, b, c, d such that a + b = d – c. So this problem can be efficiently solved using hashing.

1. Store sums of all pairs (a + b) in a hash table
2. Traverse through all pairs (c, d) again and search for (d – c) in the hash table.
3. If a pair is found with the required sum, then make sure that all elements are distinct array elements and an element is not considered more than once.

Below is the implementation in C++.

C++

```
 // A hashing based CPP program to find largest d
// such that a + b + c = d.
#include <bits/stdc++.h>
```

```
using namespace std;

// The function finds four elements with given sum X
int findFourElements(int arr[], int n)
{
    // Store sums  (a+b) of all pairs (a,b) in a
    // hash table
    unordered_map<int, pair<int, int> > mp;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            mp[arr[i] + arr[j]] = { i, j };

    // Traverse through all pairs and find (d -c)
    // is present in hash table
    int d = INT_MIN;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            int abs_diff = abs(arr[i] - arr[j]);

            // If d - c is present in hash table,
            if (mp.find(abs_diff) != mp.end()) {

                // Making sure that all elements are
                // distinct array elements and an element
                // is not considered more than once.
                pair<int, int> p = mp[abs_diff];
                if (p.first != i && p.first != j &&
                    p.second != i && p.second != j)
                    d = max(d, max(arr[i], arr[j]));
            }
        }
    }
    return d;
}

// Driver program to test above function
int main()
{
    int arr[] = { 2, 3, 5, 7, 12 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int res = findFourElements(arr, n);
    if (res == INT_MIN)
        cout << "No Solution.";
    else
        cout << res;
    return 0;
}
```

Output:

```
12
```

The overall time complexity for this efficient approach is $\mathbf{O(N^2)}$ (where N is the size of the set).

Improved By : Nishant Tanwar, Sam007, vt_m, manishshaw1

Source

https://www.geeksforgeeks.org/find-largest-d-in-array-such-that-a-b-c-d/

Chapter 101

Find lost element from a duplicated array

Find lost element from a duplicated array - GeeksforGeeks

Given two arrays which are duplicates of each other except one element, that is one element from one of the array is missing, we need to find that missing element.

Examples:

```
Input:  arr1[] = {1, 4, 5, 7, 9}
        arr2[] = {4, 5, 7, 9}
Output: 1
1 is missing from second array.

Input: arr1[] = {2, 3, 4, 5}
       arr2[] = {2, 3, 4, 5, 6}
Output: 6
6 is missing from first array.
```

One **simple solution** is to iterate over arrays and check element by element and flag the missing element when an unmatch is found, but this solution requires linear time over size of array.

Another **efficient solution** is based onbinary searchapproach. Algorithm steps are as follows:

1. Start binary search in bigger array and get mid as (lo + hi) / 2
2. If value from both array is same then missing element must be in right part so set lo as mid
3. Else set hi as mid because missing element must be in left part of bigger array if mid elements are not equal.

4. Special case are handled separately as for single element and zero element array, single element itself will be the missing element.
   If first element itself is not equal then that element will be the missing element./li>

Below is the implementation of above steps

C++

```
 // C++ program to find missing element from same
// arrays (except one missing element)
#include <bits/stdc++.h>
using namespace std;

// Funtion to find missing element based on binary
// search approach.  arr1[] is of larger size and
// N is size of it.  arr1[] and arr2[] are assumed
// to be in same order.
int findMissingUtil(int arr1[], int arr2[], int N)
{
    // special case, for only element which is
    // missing in second array
    if (N == 1)
        return arr1[0];

    // special case, for first element missing
    if (arr1[0] != arr2[0])
        return arr1[0];

    // Initialize current corner points
    int lo = 0,  hi = N - 1;

    // loop until lo < hi
    while (lo < hi)
    {
        int mid = (lo + hi) / 2;

        // If element at mid indices are equal
        // then go to right subarray
        if (arr1[mid] == arr2[mid])
            lo = mid;
        else
            hi = mid;

        // if lo, hi becomes contiguous,  break
        if (lo == hi - 1)
            break;
    }
```

```
    // missing element will be at hi index of
    // bigger array
    return arr1[hi];
}

// This function mainly does basic error checking
// and calls findMissingUtil
void findMissing(int arr1[], int arr2[], int M, int N)
{
    if (N == M-1)
        cout << "Missing Element is "
        << findMissingUtil(arr1, arr2, M) << endl;
    else if (M == N-1)
        cout << "Missing Element is "
        << findMissingUtil(arr2, arr1, N) << endl;
    else
        cout << "Invalid Input";
}

// Driver Code
int main()
{
    int arr1[] = {1, 4, 5, 7, 9};
    int arr2[] = {4, 5, 7, 9};

    int M = sizeof(arr1) / sizeof(int);
    int N = sizeof(arr2) / sizeof(int);

    findMissing(arr1, arr2, M, N);

    return 0;
}
```

Java

```
 // Java program to find missing element
// from same arrays
// (except one missing element)

import java.io.*;
class MissingNumber {

    /* Funtion to find missing element based
     on binary search approach. arr1[] is of
     larger size and N is size of it.arr1[] and
     arr2[] are assumed to be in same order. */
    int findMissingUtil(int arr1[], int arr2[],
```

```
                    int N)
{
    // special case, for only element
    // which is missing in second array
    if (N == 1)
        return arr1[0];

    // special case, for first
    // element missing
    if (arr1[0] != arr2[0])
        return arr1[0];

    // Initialize current corner points
    int lo = 0, hi = N - 1;

    // loop until lo < hi
    while (lo < hi) {
        int mid = (lo + hi) / 2;

        // If element at mid indices are
        // equal then go to right subarray
        if (arr1[mid] == arr2[mid])
            lo = mid;
        else
            hi = mid;

        // if lo, hi becomes
        // contiguous, break
        if (lo == hi - 1)
            break;
    }

    // missing element will be at hi
    // index of bigger array
    return arr1[hi];
}

// This function mainly does basic error
// checking and calls findMissingUtil
void findMissing(int arr1[], int arr2[],
                           int M, int N)
{
    if (N == M - 1)
    System.out.println("Missing Element is "
    + findMissingUtil(arr1, arr2, M) + "\n");
    else if (M == N - 1)
    System.out.println("Missing Element is "
    + findMissingUtil(arr2, arr1, N) + "\n");
```

```
        else
        System.out.println("Invalid Input");
    }

    // Driver Code
    public static void main(String args[])
    {
        MissingNumber obj = new MissingNumber();
        int arr1[] = { 1, 4, 5, 7, 9 };
        int arr2[] = { 4, 5, 7, 9 };
        int M = arr1.length;
        int N = arr2.length;
        obj.findMissing(arr1, arr2, M, N);
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # Python3 program to find missing
# element from same arrays
# (except one missing element)

# Funtion to find missing element based
# on binary search approach. arr1[] is
# of larger size and N is size of it.
# arr1[] and arr2[] are assumed
# to be in same order.
def findMissingUtil(arr1, arr2, N):

    # special case, for only element
    # which is missing in second array
    if N == 1:
        return arr1[0];

    # special case, for first
    # element missing
    if arr1[0] != arr2[0]:
        return arr1[0]

    # Initialize current corner points
    lo = 0
    hi = N - 1

    # loop until lo < hi
    while (lo < hi):
```

```
        mid = (lo + hi) / 2

        # If element at mid indices
        # are equal then go to
        # right subarray
        if arr1[mid] == arr2[mid]:
            lo = mid
        else:
            hi = mid

        # if lo, hi becomes
        # contiguous, break
        if lo == hi - 1:
            break

    # missing element will be at
    # hi index of bigger array
    return arr1[hi]

# This function mainly does basic
# error checking and calls
# findMissingUtil
def findMissing(arr1, arr2, M, N):

    if N == M-1:
        print("Missing Element is",
            findMissingUtil(arr1, arr2, M))
    elif M == N-1:
        print("Missing Element is",
            findMissingUtil(arr2, arr1, N))
    else:
        print("Invalid Input")

# Driver Code
arr1 = [1, 4, 5, 7, 9]
arr2 = [4, 5, 7, 9]
M = len(arr1)
N = len(arr2)
findMissing(arr1, arr2, M, N)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find missing element from
// same arrays (except one missing element)
using System;
```

```
class GFG {

    /* Funtion to find missing element based
    on binary search approach. arr1[] is of
    larger size and N is size of it.arr1[] and
    arr2[] are assumed to be in same order. */
    static int findMissingUtil(int []arr1,
                           int []arr2, int N)
    {

        // special case, for only element
        // which is missing in second array
        if (N == 1)
            return arr1[0];

        // special case, for first
        // element missing
        if (arr1[0] != arr2[0])
            return arr1[0];

        // Initialize current corner points
        int lo = 0, hi = N - 1;

        // loop until lo < hi
        while (lo < hi) {
            int mid = (lo + hi) / 2;

            // If element at mid indices are
            // equal then go to right subarray
            if (arr1[mid] == arr2[mid])
                lo = mid;
            else
                hi = mid;

            // if lo, hi becomes
            // contiguous, break
            if (lo == hi - 1)
                break;
        }

        // missing element will be at hi
        // index of bigger array
        return arr1[hi];
    }

    // This function mainly does basic error
    // checking and calls findMissingUtil
    static void findMissing(int []arr1, int []arr2,
```

```
                            int M, int N)
    {
        if (N == M - 1)
            Console.WriteLine("Missing Element is "
            + findMissingUtil(arr1, arr2, M) + "\n");
        else if (M == N - 1)
            Console.WriteLine("Missing Element is "
            + findMissingUtil(arr2, arr1, N) + "\n");
        else
            Console.WriteLine("Invalid Input");
    }

    // Driver Code
    public static void Main()
    {
        int []arr1 = { 1, 4, 5, 7, 9 };
        int []arr2 = { 4, 5, 7, 9 };
        int M = arr1.Length;
        int N = arr2.Length;
        findMissing(arr1, arr2, M, N);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find missing
// element from same arrays
// (except one missing element)

// Function to find missing
// element based on binary
// search approach. arr1[]
// is of larger size and
// N is size of it. arr1[]
// and arr2[] are assumed
// to be in same order.
function findMissingUtil($arr1, $arr2, $N)
{

    // special case, for only
    // element which is
    // missing in second array
    if ($N == 1)
        return $arr1[0];
```

```
    // special case, for first
    // element missing
    if ($arr1[0] != $arr2[0])
        return $arr1[0];

    // Initialize current
    // corner points
    $lo = 0;
    $hi = $N - 1;

    // loop until lo < hi
    while ($lo < $hi)
    {
        $mid = ($lo + $hi) / 2;

        // If element at mid indices are
        // equal then go to right subarray
        if ($arr1[$mid] == $arr2[$mid])
            $lo = $mid;
        else
            $hi = $mid;

        // if lo, hi becomes
        // contiguous, break
        if ($lo == $hi - 1)
            break;
    }

    // missing element will be
    // at hi index of
    // bigger array
    return $arr1[$hi];
}

// This function mainly
// does basic error checking
// and calls findMissingUtil
function findMissing($arr1, $arr2,
                            $M, $N)
{
    if ($N == $M - 1)
        echo "Missing Element is "
             , findMissingUtil($arr1,
                         $arr2, $M) ;
    else if ($M == $N - 1)
        echo "Missing Element is "
             , findMissingUtil($arr2,
                            $arr1, $N);
```

```
    else
        echo "Invalid Input";
}

    // Driver Code
    $arr1 = array(1, 4, 5, 7, 9);
    $arr2 = array(4, 5, 7, 9);
    $M = count($arr1);
    $N = count($arr2);
    findMissing($arr1, $arr2, $M, $N);

// This code is contributed by anuj_67.
?>
```

Output :

```
Missing Element is 1
```

What if input arrays are not in same order?
In this case, missing element is simply XOR of all elements of both arrays. Thanks to Yolo Song for suggesting this.

CPP

```
 // C++ program to find missing element from one array
// such that it has all elements of other array except
// one.  Elements in two arrays can be in any order.
#include <bits/stdc++.h>
using namespace std;

// This function mainly does XOR of all elements
// of arr1[] and arr2[]
void findMissing(int arr1[], int arr2[], int M,
                 int N)
{
    if (M != N-1 && N != M-1)
    {
        cout << "Invalid Input";
        return;
    }

    // Do XOR of all element
    int res = 0;
    for (int i=0; i<M; i++)
       res = res^arr1[i];
    for (int i=0; i<N; i++)
```

```
       res = res^arr2[i];

    cout << "Missing element is " << res;
}

// Driver Code
int main()
{
    int arr1[] = {4, 1, 5, 9, 7};
    int arr2[] = {7, 5, 9, 4};

    int M = sizeof(arr1) / sizeof(int);
    int N = sizeof(arr2) / sizeof(int);

    findMissing(arr1, arr2, M, N);

    return 0;
}
```

Java

```
 // Java program to find missing element
// from one array such that it has all
// elements of other array except one.
// Elements in two arrays can be in any order.

import java.io.*;
class Missing {

    // This function mainly does XOR of
    // all elements of arr1[] and arr2[]
    void findMissing(int arr1[], int arr2[],
                               int M, int N)
    {
        if (M != N - 1 && N != M - 1) {
        System.out.println("Invalid Input");
            return;
        }

        // Do XOR of all element
        int res = 0;
        for (int i = 0; i < M; i++)
            res = res ^ arr1[i];
        for (int i = 0; i < N; i++)
            res = res ^ arr2[i];

        System.out.println("Missing element is "
                                          + res);
```

```
    }

    // Driver Code
    public static void main(String args[])
    {
        Missing obj = new Missing();
        int arr1[] = { 4, 1, 5, 9, 7 };
        int arr2[] = { 7, 5, 9, 4 };
        int M = arr1.length;
        int N = arr2.length;
        obj.findMissing(arr1, arr2, M, N);
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # Python 3 program to find
# missing element from one array
# such that it has all elements
# of other array except
# one. Elements in two arrays
# can be in any order.

# This function mainly does XOR of all elements
# of arr1[] and arr2[]
def findMissing(arr1,arr2, M, N):
    if (M != N-1 and N != M-1):

        print("Invalid Input")
        return


    # Do XOR of all element
    res = 0
    for i in range(0,M):
        res = res^arr1[i];
    for i in range(0,N):
        res = res^arr2[i]

    print("Missing element is",res)

# Driver Code
arr1 = [4, 1, 5, 9, 7]
arr2 = [7, 5, 9, 4]
M = len(arr1)
N = len(arr2)
```

```
findMissing(arr1, arr2, M, N)

# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# program to find missing element
// from one array such that it has all
// elements of other array except one.
// Elements in two arrays can be in
// any order.
using System;
class GFG {

    // This function mainly does XOR of
    // all elements of arr1[] and arr2[]
    static void findMissing(int []arr1,
                            int []arr2,
                            int M, int N)
    {
        if (M != N - 1 && N != M - 1)
        {
            Console.WriteLine("Invalid Input");
            return;
        }

        // Do XOR of all element
        int res = 0;
        for (int i = 0; i < M; i++)
            res = res ^ arr1[i];
        for (int i = 0; i < N; i++)
            res = res ^ arr2[i];

        Console.WriteLine("Missing element is "
                                        + res);
    }

    // Driver Code
    public static void Main()
    {

        int []arr1 = {4, 1, 5, 9, 7};
        int []arr2 = {7, 5, 9, 4};
        int M = arr1.Length;
        int N = arr2.Length;
        findMissing(arr1, arr2, M, N);
    }
```

```
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find missing
// element from one array
// such that it has all elements
// of other array except
// one. Elements in two arrays
// can be in any order.

// This function mainly does
// XOR of all elements
// of arr1[] and arr2[]
function findMissing($arr1, $arr2,
                            $M, $N)
{
    if ($M != $N - 1 && $N != $M - 1)
    {
        echo "Invalid Input";
        return;
    }

    // Do XOR of all element
    $res = 0;
    for ($i = 0; $i < $M; $i++)
        $res = $res ^ $arr1[$i];

    for ($i = 0; $i < $N; $i++)
    $res = $res ^ $arr2[$i];
    echo "Missing element is ", $res;
}

    // Driver Code
    $arr1 = array(4, 1, 5, 9, 7);
    $arr2 = array(7, 5, 9, 4);
    $M = sizeof($arr1);
    $N = sizeof($arr2);

    findMissing($arr1, $arr2, $M, $N);

// This code is contributed by aj_36
?>
```

Output :

```
Missing Element is 1
```

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/find-lost-element-from-a-duplicated-array/

Chapter 102

Find missing elements of a range

Find missing elements of a range - GeeksforGeeks

Given an array arr[0..n-1] of distinct elements and a range [low, high], find all numbers that are in range, but not in array. The missing elements should be printed in sorted order.

Examples:

```
Input: arr[] = {10, 12, 11, 15},
       low = 10, hight = 15
Output: 13, 14

Input: arr[] = {1, 14, 11, 51, 15},
       low = 50, hight = 55
Output: 50, 52, 53, 54
```

There can be following two approaches to solve the problem.

1. **Use Sorting :** Sort the array, then do binary search for 'low'. Once location of low is find, start traversing array from that location and keep printing all missing numbers.

   C++

   ```
    // A sorting based C++ program to find missing
   // elements from an array
   #include<bits/stdc++.h>
   using namespace std;

   // Print all elements of range [low, high] that
   ```

```
// are not present in arr[0..n-1]
void printMissing(int arr[], int n, int low,
                                    int high)
{
   // Sort the array
   sort(arr, arr+n);

   // Do binary search for 'low' in sorted
   // array and find index of first element
   // which either equal to or greater than
   // low.
   int *ptr = lower_bound(arr, arr+n, low);
   int index = ptr - arr;

   // Start from the found index and linearly
   // search every range element x after this
   // index in arr[]
   int i = index, x = low;
   while (i < n && x<=high)
   {
       // If x doesn't math with current element
       // print it
       if (arr[i] != x)
          cout << x << " ";

       // If x matches, move to next element in arr[]
       else
          i++;

       // Move to next element in range [low, high]
       x++;
   }

   // Print range elements thar are greater than the
   // last element of sorted array.
   while (x <= high)
     cout << x++ << " ";
}

// Driver program
int main()
{
   int arr[] = {1, 3, 5, 4};
   int n = sizeof(arr)/sizeof(arr[0]);
   int low = 1, high = 10;
   printMissing(arr, n, low, high);
   return 0;
}
```

Java

```
 // A sorting based Java program to find missing
// elements from an array

import java.util.Arrays;

public class PrintMissing
{
    // Print all elements of range [low, high] that
    // are not present in arr[0..n-1]
    static void printMissing(int ar[], int low, int high)
    {
        Arrays.sort(ar);
        // Do binary search for 'low' in sorted
        // array and find index of first element
        // which either equal to or greater than
        // low.
        int index = ceilindex(ar, low, 0, ar.length - 1);
        int x = low;

        // Start from the found index and linearly
        // search every range element x after this
        // index in arr[]
        while (index < ar.length && x <= high)
        {
            // If x doesn't math with current element
            // print it
            if (ar[index] != x)
            {
                System.out.print(x + " ");
            }

            // If x matches, move to next element in arr[]
            else
                index++;
            // Move to next element in range [low, high]
            x++;
        }

        // Print range elements thar are greater than the
        // last element of sorted array.
        while (x <= high)
        {
            System.out.print(x + " ");
            x++;
        }
```

```
    }

    // Utility function to find ceil index of given element
    static int ceilindex(int ar[], int val, int low, int high)
    {

        if (val < ar[0])
            return 0;
        if (val > ar[ar.length - 1])
            return ar.length;

        int mid = (low + high) / 2;
        if (ar[mid] == val)
            return mid;
        if (ar[mid] < val)
        {
            if (mid + 1 < high && ar[mid + 1] >= val)
                return mid + 1;
            return ceilindex(ar, val, mid + 1, high);
        }
        else
        {
            if (mid - 1 >= low && ar[mid - 1] < val)
                return mid;
            return ceilindex(ar, val, low, mid - 1);
        }

    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int arr[] = { 1, 3, 5, 4 };
        int low = 1, high = 10;
        printMissing(arr, low, high);
    }
}

// This code is contributed by Rishabh Mahrsee
```

Output:

```
2 6 7 8 9 10
```

2. **Use Hashing :** Create a hash table and insert all array items into the hash table. Once all items are in hash table, traverse through the range and print all missing elements.

C++

```
 // A hashing based C++ program to find missing
// elements from an array
#include<bits/stdc++.h>
using namespace std;

// Print all elements of range [low, high] that
// are not present in arr[0..n-1]
void printMissing(int arr[], int n, int low,
                                    int high)
{
   // Insert all elements of arr[] in set
   unordered_set<int> s;
   for (int i=0; i<n; i++)
      s.insert(arr[i]);

   // Traverse throught the range an print all
   // missing elements
   for (int x=low; x<=high; x++)
      if (s.find(x) == s.end())
         cout << x << " ";
}

// Driver program
int main()
{
   int arr[] = {1, 3, 5, 4};
   int n = sizeof(arr)/sizeof(arr[0]);
   int low = 1, high = 10;
   printMissing(arr, n, low, high);
   return 0;
}
```

Java

```
 // A hashing based Java program to find missing
// elements from an array

import java.util.Arrays;
import java.util.HashSet;

public class Print
{
    // Print all elements of range [low, high] that
    // are not present in arr[0..n-1]
    static void printMissing(int ar[], int low, int high)
    {
        HashSet<Integer> hs = new HashSet<>();
```

```
            // Insert all elements of arr[] in set
            for (int i = 0; i < ar.length; i++)
                hs.add(ar[i]);

            // Traverse throught the range an print all
            // missing elements
            for (int i = low; i <= high; i++)
            {
                if (!hs.contains(i))
                {
                    System.out.print(i + " ");
                }
            }
        }

        // Driver program to test above function
        public static void main(String[] args)
        {
            int arr[] = { 1, 3, 5, 4 };
            int low = 1, high = 10;
            printMissing(arr, low, high);
        }
    }

    // This code is contributed by Rishabh Mahrsee
```

Output:

```
2 6 7 8 9 10
```

Which approach is better?
Time complexity of first approach is O(nLogn + k) where k is number of missing elements (Note that k may be more than nLogn if array is small and range is big)

Time complexity of second solution is O(n + (high-low+1)).

If the given array has almost elements of range i.e., n is close to value of (high-low+1), then second approach is definitely better as there is no Log n factor. But if n is much smaller than range, then first approach is better as it doesn't require extra space for hashing. We can also modify first approach to print adjacent missing elements as range to save time. For example if 50, 51, 52, 53, 54, 59 are missing, we can print them as 50-54, 59 in first method. And if printing this way is allowed, the first approach takes only O(n Log n) time.

This article is contributed by **Piyush Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/find-missing-elements-of-a-range/

Chapter 103

Find number of Employees Under every Employee

Find number of Employees Under every Employee - GeeksforGeeks

Given a dictionary that contains mapping of employee and his manager as a number of (employee, manager) pairs like below.

```
{ "A", "C" },
{ "B", "C" },
{ "C", "F" },
{ "D", "E" },
{ "E", "F" },
{ "F", "F" }

In this example C is manager of A,
C is also manager of B, F is manager
of C and so on.
```

Write a function to get no of employees under each manager in the hierarchy not just their direct reports. It may be assumed that an employee directly reports to only one manager. In the above dictionary the root node/ceo is listed as reporting to himself.

Output should be a Dictionary that contains following.

```
A - 0
B - 0
C - 2
D - 0
E - 1
F - 5
```

Source: Microsoft Interview

This question might be solved differently but i followed this and found interesting, so sharing:

```
 1. Create a reverse map with Manager->DirectReportingEmployee
    combination. Off-course employee will be multiple so Value
    in Map is List of Strings.
        "C" --> "A", "B",
        "E" --> "D"
        "F" --> "C", "E", "F"


2. Now use the given employee-manager map to iterate  and at
   the same time use newly reverse map to find the count of
   employees under manager.

   Let the map created in step 2 be 'mngrEmpMap'
   Do following for every employee 'emp'.
     a) If 'emp' is not present in 'mngrEmpMap'
          Count under 'emp' is 0 [Nobody reports to 'emp']
     b) If 'emp' is present in 'mngrEmpMap'
          Use the list of direct reports from map 'mngrEmpMap'
          and recursively calculate number of total employees
          under 'emp'.
```

A trick in step 2.b is to use memorization(Dynamic programming) while finding number of employees under a manager so that we don't need to find number of employees again for any of the employees. In the below code populateResultUtil() is the recursive function that uses memoization to avoid re-computation of same results.

Below is Java implementation of above ides

```
 // Java program to find number of persons under every employee
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class NumberEmployeeUnderManager
{
    // A hashmap to store result. It stores count of employees
    // under every employee, the count may by 0 also
    static Map<String,Integer> result =
                              new HashMap<String, Integer>();

    // Driver function
    public static void main(String[] args)
```

```
{
    Map<String, String> dataSet = new HashMap<String, String>();
    dataSet.put("A", "C");
    dataSet.put("B", "C");
    dataSet.put("C", "F");
    dataSet.put("D", "E");
    dataSet.put("E", "F");
    dataSet.put("F", "F");

    populateResult(dataSet);
    System.out.println("result = " + result);
}

// This function populates 'result' for given input 'dataset'
private static void populateResult(Map<String, String> dataSet)
{
    // To store reverse of original map, each key will have 0
    // to multiple values
    Map<String, List<String>> mngrEmpMap =
                              new HashMap<String, List<String>>();

    // To fill mngrEmpMap, iterate through the given map
    for (Map.Entry<String,String> entry: dataSet.entrySet())
    {
        String emp = entry.getKey();
        String mngr = entry.getValue();
        if (!emp.equals(mngr)) // excluding emp-emp entry
        {
            // Get the previous list of direct reports under
            // current 'mgr' and add the current 'emp' to the list
            List<String> directReportList = mngrEmpMap.get(mngr);

            // If 'emp' is the first employee under 'mgr'
            if (directReportList == null)
                directReportList = new ArrayList<String>();

            directReportList.add(emp);

            // Replace old value for 'mgr' with new
            // directReportList
            mngrEmpMap.put(mngr, directReportList);
        }
    }

    // Now use manager-Emp map built above to populate result
    // with use of populateResultUtil()

    // note- we are iterating over original emp-manager map and
```

```
        // will use mngr-emp map in helper to get the count
        for (String mngr: dataSet.keySet())
            populateResultUtil(mngr, mngrEmpMap);
    }

    // This is a recursive function to fill count for 'mgr' using
    // mngrEmpMap.  This function uses memoization to avoid re-
    // computations of subproblems.
    private static int populateResultUtil(String mngr,
                                Map<String, List<String>> mngrEmpMap)
    {
        int count = 0;

        // means employee is not a manager of any other employee
        if (!mngrEmpMap.containsKey(mngr))
        {
            result.put(mngr, 0);
            return 0;
        }

        // this employee count has already been done by this
        // method, so avoid re-computation
        else if (result.containsKey(mngr))
            count = result.get(mngr);

        else
        {
            List<String> directReportEmpList = mngrEmpMap.get(mngr);
            count = directReportEmpList.size();
            for (String directReportEmp: directReportEmpList)
               count +=  populateResultUtil(directReportEmp, mngrEmpMap);

            result.put(mngr, count);
        }
        return count;
    }
}
```

Output:

```
result = {D=0, E=1, F=5, A=0, B=0, C=2}
```

This article is contributed by **Chandan Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/find-number-of-employees-under-every-manager/

Chapter 104

Find number of pairs in an array such that their XOR is 0

Find number of pairs in an array such that their XOR is 0 - GeeksforGeeks

Given an array $A[]$ of size N. Find the number of pairs (i, j) such that A_i XOR A_j = 0, and 1 <= i < j <= N.

Examples :

```
Input : A[] = {1, 3, 4, 1, 4}
Output : 2
Explanation : Index (0, 3) and (2, 4)

Input : A[] = {2, 2, 2}
Output : 3
```

First Approach : **Sorting**

A_i XOR A_j = 0 is only satisfied when $A_i = A_j$. Therefore, we will first sort the array and then count the frequency of each element. By combinatorics, we can observe that if frequency of some element is $count$ then, it will contribute $count * (count - 1) / 2$ to the answer.

Below is the implementation of above approach :

C++

```
// C++ program to find number
```

```
// of pairs in an array such
// that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the
// count
int calculate(int a[], int n)
{
    // Sorting the list using
    // built in function
    sort(a, a + n);

    int count = 1;
    int answer = 0;

    // Traversing through the
    // elements
    for (int i = 1; i < n; i++) {

        if (a[i] == a[i - 1]){

            // Counting frequency of each
            // elements
            count += 1;

        }
        else
        {
            // Adding the contribution of
            // the frequency to the answer
            answer = answer + (count * (count - 1)) / 2;
            count = 1;
        }
    }

    answer = answer + (count * (count - 1)) / 2;

    return answer;
}

// Driver Code
int main()
{

    int a[] = { 1, 2, 1, 2, 4 };
    int n = sizeof(a) / sizeof(a[0]);
```

```
    // Print the count
    cout << calculate(a, n);
    return 0;
}

// This article is contributed by Sahil_Bansall.
```

Java

```
 // Java program to find number
// of pairs in an array such
// that their XOR is 0
import java.util.*;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int a[], int n)
    {
        // Sorting the list using
        // built in function
        Arrays.sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }
```

```
        answer = answer + (count * (count - 1)) / 2;

        return answer;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int a[] = { 1, 2, 1, 2, 4 };
        int n = a.length;

        // Print the count
        System.out.println(calculate(a, n));
    }
}

// This code is contributed by Ansu Kumari.
```

Python3

```
 # Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the count
def calculate(a) :

    # Sorting the list using
    # built in function
    a.sort()

    count = 1
    answer = 0

    # Traversing through the elements
    for i in range(1, len(a)) :

        if a[i] == a[i - 1] :

            # Counting frequncy of each elements
            count += 1

        else :

            # Adding the contribution of
            # the frequency to the answer
            answer = answer + count * (count - 1) // 2
            count = 1
```

```
    answer = answer + count * (count - 1) // 2

    return answer


# Driver Code
if __name__ == '__main__':

    a = [1, 2, 1, 2, 4]

    # Print the count
    print(calculate(a))
```

C#

```
 // Java program to find number
// of pairs in an array such
// that their XOR is 0
using System;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int []a, int n)
    {
        // Sorting the list using
        // built in function
        Array.Sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
```

```
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }

        answer = answer + (count * (count - 1)) / 2;

        return answer;
    }

    // Driver Code
    public static void Main ()
    {
        int []a = { 1, 2, 1, 2, 4 };
        int n = a.Length;

        // Print the count
        Console.WriteLine(calculate(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find number
// of pairs in an array such
// that their XOR is 0

// Function to calculate
// the count
function calculate($a, $n)
{

    // Sorting the list using
    // built in function
    sort($a);

    $count = 1;
    $answer = 0;

    // Traversing through the
    // elements
    for ($i = 1; $i < $n; $i++)
    {
```

```
        if ($a[$i] == $a[$i - 1])
        {

            // Counting frequency of
            // each elements
            $count += 1;

        }

        else
        {

            // Adding the contribution of
            // the frequency to the answer
            $answer = $answer + ($count *
                        ($count - 1)) / 2;
            $count = 1;
        }
    }

    $answer = $answer + ($count *
                ($count - 1)) / 2;

    return $answer;
}

    // Driver Code
    $a = array(1, 2, 1, 2, 4);
    $n = count($a);

    // Print the count
    echo calculate($a, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
2
```

Time Complexity : O(N Log N)

Second Approach : **Hashing (Index Mapping)**

Solution is handy, if we can count the frequency of each element in the array. Index mapping technique can be used to count the frequency of each element.

Below is the implementation of above approach :

C++

```
 // C++ program to find number of pairs
// in an array such that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the answer
int calculate(int a[]){

    // Finding the maximum of the array
    int *maximum = max_element(a, a + 5);

    // Creating frequency array
    // With initial value 0
    int frequency[*maximum + 1] = {0};

    // Traversing through the array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Counting frequency
        frequency[a[i]] += 1;
    }
    int answer = 0;

    // Traversing through the frequency array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Calculating answer
        answer = answer + frequency[i] * (frequency[i] - 1) ;
    }
    return answer/2;
}

// Driver Code
int main()
{
   int a[] = {1, 2, 1, 2, 4};

   // Function calling
   cout << (calculate(a));
}

// This code is contributed by Smitha
```

Python 3

```
 # Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the answer
def calculate(a) :

    # Finding the maximum of the array
    maximum = max(a)

    # Creating frequency array
    # With initial value 0
    frequency = [0 for x in range(maximum + 1)]

    # Traversing through the array
    for i in a :

        # Counting frequency
        frequency[i] += 1

    answer = 0

    # Traversing through the frequency array
    for i in frequency :

        # Calculating answer
        answer = answer + i * (i - 1) // 2

    return answer

# Driver Code
a = [1, 2, 1, 2, 4]
print(calculate(a))
```

PHP

Output :

```
2
```

Time Complexity : O(N)

Note : Index Mapping method can only be used when the numbers in the array are not large. In such cases, sorting method can be used.

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-number-pairs-array-xor-0/

Chapter 105

Find pair with greatest product in array

Find pair with greatest product in array - GeeksforGeeks

Given an array of n elements, the task is to find the greatest number such that it is product of two elements of given array. If no such element exists, print -1. Elements are within the range of 1 to 10^5.

Examples :

```
Input :  arr[] = {10, 3, 5, 30, 35}
Output:  30
Explanation: 30 is the product of 10 and 3.

Input :  arr[] = {2, 5, 7, 8}
Output:  -1
Explanation: Since, no such element exists.

Input :  arr[] = {10, 2, 4, 30, 35}
Output:  -1

Input :  arr[] = {10, 2, 2, 4, 30, 35}
Output:  4

Input  : arr[] = {17, 2, 1, 35, 30}
Output : 35
```

A **naive approach** is to pick an element and then check for each pair product if equal to that number and update the max if the number is maximum,repeat until whole array gets

traversed takes O(n^3) time.

C++

```
 // C++ program to find a pair with product
// in given array.
#include<bits/stdc++.h>
using namespace std;

// Function to find greatest number that us
int findGreatest( int arr[] , int n)
{
    int result = -1;
    for (int i = 0; i < n ; i++)
        for (int j = 0; j < n-1; j++)
            for (int k = j+1 ; k < n  ; k++)
                if (arr[j] * arr[k] == arr[i])
                    result = max(result, arr[i]);
    return result;
}

// Driver code
int main()
{
    // Your C++ Code
    int arr[] = {30, 10, 9, 3, 35};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << findGreatest(arr, n);

    return 0;
}
```

Java

```
 // Java program to find a pair
// with product in given array.
import java.io.*;

class GFG{

static int findGreatest( int []arr , int n)
{
    int result = -1;
    for (int i = 0; i < n ; i++)
        for (int j = 0; j < n-1; j++)
            for (int k = j+1 ; k < n ; k++)
```

```
                if (arr[j] * arr[k] == arr[i])
                    result = Math.max(result, arr[i]);
    return result;
}

    // Driver code
    static public void main (String[] args)
    {
        int []arr = {30, 10, 9, 3, 35};
        int n = arr.length;

        System.out.println(findGreatest(arr, n));
    }
}

//This code is contributed by vt_m.
```

C#

```
 // C# program to find a pair with product
// in given array.
using System;

class GFG{

static int findGreatest( int []arr , int n)
{
    int result = -1;
    for (int i = 0; i < n ; i++)
        for (int j = 0; j < n-1; j++)
            for (int k = j+1 ; k < n ; k++)
                if (arr[j] * arr[k] == arr[i])
                    result = Math.Max(result, arr[i]);
    return result;
}

    // Driver code
    static public void Main ()
    {
       int []arr = {30, 10, 9, 3, 35};
       int n = arr.Length;

       Console.WriteLine(findGreatest(arr, n));
    }
}

//This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find a pair
// with product in given array.

// Function to find
// greatest number
function findGreatest($arr , $n)
{
    $result = -1;
    for ($i = 0; $i < $n ; $i++)
        for ($j = 0; $j < $n-1; $j++)
            for ($k = $j+1 ; $k < $n ; $k++)
                if ($arr[$j] * $arr[$k] == $arr[$i])
                    $result = max($result, $arr[$i]);
    return $result;
}

// Driver code
$arr = array(30, 10, 9, 3, 35);
$n = count($arr);

echo findGreatest($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
30
```

An **efficient method** follows below implementation:-

1. Create an empty hash table and store all array elements in it.
2. Sort the array in ascending order.
3. Pick elements one by one from end of the array.
4. And check if there exists a pair whose product is equal to that number. In this efficiency can be achieved. The idea is to reach till sqrt of that number. If we don't get the pair till sqrt that means no such pair exists. We use hash table to make sure that we can find other element of pair in O(1) time.
5. Repeat steps 2 to 3 until we get the element or whole array gets traversed.

Below is C++ implementation.

C++

```
 // C++ program to find the largest product number
#include<bits/stdc++.h>
using namespace std;

// Function to find greatest number
int findGreatest(int arr[], int n)
{
    // Store occurrences of all elements in hash
    // array
    unordered_map<int, int> m;
    for (int i = 0 ; i < n; i++)
        m[arr[i]]++;

    // Sort the array and traverse all elements from
    // end.
    sort(arr, arr+n);

    for (int i=n-1; i>1; i--)
    {
        // For every element, check if there is another
        // element which divides it.
        for (int j=0; j<i && arr[j]<=sqrt(arr[i]); j++)
        {
            if (arr[i] % arr[j] == 0)
            {
                int result = arr[i]/arr[j];

                // Check if the result value exists in array
                // or not if yes the return arr[i]
                if (result != arr[j] && m[result] > 0)
                    return arr[i];

                // To handle the case like arr[i] = 4 and
                // arr[j] = 2
                else if (result == arr[j] && m[result] > 1)
                    return arr[i];
            }
        }
    }
    return -1;
}

// Drivers code
int main()
{
    int arr[] = {17, 2, 1, 15, 30};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << findGreatest(arr, n);
```

```
    return 0;
}
```

Output :

```
30
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-pair-with-greatest-product-in-array/

Chapter 106

Find pairs in array whose sums already exist in array

Find pairs in array whose sums already exist in array - GeeksforGeeks

Given an array of n distinct and positive elements, the task is to find pair whose sum already exists in given array.

Examples :

```
Input : arr[] = {2, 8, 7, 1, 5};
Output : 2 5
         7 1

Input : arr[] = {7, 8, 5, 9, 11};
Output : Not Exist
```

A **Naive Approach** is to run three loops to find pair whose sum exists in array.

C++

```
 // A simple C++ program to find pair whose sum
// already exists in array
#include <bits/stdc++.h>
using namespace std;

// Function to find pair whose sum exists in arr[]
void findPair(int arr[], int n)
{
    bool found = false;
    for (int i = 0; i < n; i++) {
```

```
        for (int j = i + 1; j < n; j++) {
            for (int k = 0; k < n; k++) {
                if (arr[i] + arr[j] == arr[k]) {
                    cout << arr[i] << " " << arr[j] << endl;
                    found = true;
                }
            }
        }
    }

    if (found == false)
        cout << "Not exist" << endl;
}

// Driven code
int main()
{
    int arr[] = { 10, 4, 8, 13, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findPair(arr, n);
    return 0;
}
```

Java

```
 // A simple Java program to find
// pair whose sum already exists
// in array
import java.io.*;

public class GFG {

    // Function to find pair whose
    // sum exists in arr[]
    static void findPair(int[] arr, int n)
    {
        boolean found = false;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    if (arr[i] + arr[j] == arr[k]) {
                        System.out.println(arr[i] +
                                           " " + arr[j]);
                        found = true;
                    }
                }
            }
        }
```

```
        if (found == false)
            System.out.println("Not exist");
    }

    // Driver code
    static public void main(String[] args)
    {
        int[] arr = {10, 4, 8, 13, 5};
        int n = arr.length;
        findPair(arr, n);
    }
}

// This code is contributed by vt_m.
```

C#

```
 // A simple C# program to find
// pair whose sum already exists
// in array
using System;

public class GFG {

    // Function to find pair whose
    // sum exists in arr[]
    static void findPair(int[] arr, int n)
    {
        bool found = false;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    if (arr[i] + arr[j] == arr[k]) {
                        Console.WriteLine(arr[i] +
                                          " " + arr[j]);
                        found = true;
                    }
                }
            }
        }

        if (found == false)
            Console.WriteLine("Not exist");
    }

    // Driver code
    static public void Main(String []args)
```

```
    {
        int[] arr = {10, 4, 8, 13, 5};
        int n = arr.Length;
        findPair(arr, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// A simple php program to
// find pair whose sum
// already exists in array

// Function to find pair whose
// sum exists in arr[]
function findPair($arr, $n)
{
    $found = false;
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = $i + 1; $j < $n; $j++)
        {
            for ($k = 0; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] == $arr[$k])
                {
                    echo $arr[$i] , " " , $arr[$j] ;
                    $found = true;
                }
            }
        }
    }

    if ($found == false)
        echo "Not exist" ;
}

// Driver code
$arr = array( 10, 4, 8, 13, 5 );
$n = sizeof($arr) / sizeof($arr[0]);
findPair($arr, $n);

// This code is contributed by nitin mittal.
?>
```

Output :

```
8 5
```

An **Efficient solution** is to store all element in an hash table (unordered_set in C++) and check one by one all pairs and check its sum exists in set or not. If it exists in set then print pair. If no pair found in array then print not exists .

C++

```
 // C++ program to find pair whose sum already
// exists in array
#include <bits/stdc++.h>
using namespace std;

// Function to find pair whose sum exists in arr[]
void findPair(int arr[], int n)
{
    // Hash to store all element of array
    unordered_set<int> s;
    for (int i = 0; i < n; i++)
        s.insert(arr[i]);

    bool found = false;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            // Check sum already exists or not
            if (s.find(arr[i] + arr[j]) != s.end()) {
                cout << arr[i] << " " << arr[j] << endl;
                found = true;
            }
        }
    }

    if (found == false)
        cout << "Not exist" << endl;
}

// Driven code
int main()
{
    int arr[] = { 10, 4, 8, 13, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findPair(arr, n);
    return 0;
}
```

Java

```
 // Java program to find pair whose sum
// already exists in array
import java.util.*;
import java.lang.*;
import java.io.*;

class Getpairs {
    // Function to find pair whose sum
    // exists in arr[]
    public static void findPair(int[] arr, int n)
    {
        /* store all the array elements as a
        Hash value*/
        HashSet<Integer> s = new HashSet<Integer>();

        for (Integer i : arr) {
            s.add(i);
        }

        /* Run two loop and check for the sum
    in the Hashset*/
        /* if not a single pair exist then found
    will be false else true*/
        boolean found = false;

        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                int sum = arr[i] + arr[j];
                if (s.contains(sum)) {
                    /* if the sum is present in
                 hashset then found become
                true*/
                    found = true;

                    System.out.println(arr[i] + " "
                                       + arr[j]);
                }
            }
        }

        if (found == false)
            System.out.println("Not Exist ");
    }

    // driver function
    public static void main(String args[])
```

```
    {
        int[] arr = { 10, 4, 8, 13, 5 };
        int n = arr.length;
        findPair(arr, n);
    }
}

// This code is contributed by Smarak Chopdar
```

Output:

```
8 5
```

Time Complexity : $O(n^2)$

Improved By : vt_m, nitin mittal

Source

https://www.geeksforgeeks.org/find-pairs-in-array-whose-sums-already-exist-in-array/

Chapter 107

Find pairs with given sum such that elements of pair are in different rows

Find pairs with given sum such that elements of pair are in different rows - GeeksforGeeks

Given a matrix of distinct values and a sum. The task is to find all the pairs in given whose summation is equal to given sum. Each element of pair must be from different rows i.e; pair must not lie in same row.

Examples:

```
Input : mat[4][4] = {{1, 3, 2, 4},
                     {5, 8, 7, 6},
                     {9, 10, 13, 11},
                     {12, 0, 14, 15}}
        sum = 11
Output: (1, 10), (3, 8), (2, 9), (4, 7), (11, 0)
```

Method 1 (Simple)

A simple solution for this problem is to one by one take each element of all rows and find pair starting from immediate next row with in matrix. Time complexity for this approach will be O(n^4).

Method 2 (Use Sorting)

- Sort all the rows in ascending order. Time complexity for this preprocessing will be O(n^2 logn).

- Now we will select each row one by one and find pair element in remaining rows after current row.
- Take two iterators **left** and **right**. **left** iterator points left corner of current i'th row and **right** iterator points right corner of next j'th row in which we are going to find pair element.
- If **mat**[**i**][**left**] + **mat**[**j**][**right**] < **sum** then **left**++ i.e; move in i'th row towards right corner, otherwise **right**++ i.e; move in j'th row towards left corner.

CPP

```
 // C++ program to find a pair with given sum such that
// every element of pair is in different rows.
#include<bits/stdc++.h>
using namespace std;

const int MAX = 100;

// Function to find pair for given sum in matrix
// mat[][] --> given matrix
// n --> order of matrix
// sum --> given sum for which we need to find pair
void pairSum(int mat[][MAX], int n, int sum)
{
    // First sort all the rows in ascending order
    for (int i=0; i<n; i++)
        sort(mat[i], mat[i]+n);

    // Select i'th row and find pair for element in i'th
    // row in j'th row whose summation is equal to given sum
    for (int i=0; i<n-1; i++)
    {
        for (int j=i+1; j<n; j++)
        {
            int left = 0, right = n-1;
            while (left<n && right>=0)
            {
                if ((mat[i][left] + mat[j][right]) == sum)
                {
                    cout << "(" << mat[i][left]
                         << ", "<< mat[j][right] << "), ";
                    left++;
                    right--;
                }
                else
                {
                    if ((mat[i][left] + mat[j][right]) < sum)
                        left++;
                    else
```

```
                        right--;
                }
            }
        }
    }
}

// Driver program to run the case
int main()
{
    int n = 4, sum = 11;
    int mat[][MAX] = {{1, 3, 2, 4},
                      {5, 8, 7, 6},
                      {9, 10, 13, 11},
                      {12, 0, 14, 15}};
    pairSum(mat, n, sum);
    return 0;
}
```

Java

```
 // Java program to find a pair with
// given sum such that every element
// of pair is in different rows.
import java.util.Arrays;
class GFG {
static final int MAX = 100;

// Function to find pair for given sum in
// matrix mat[][] --> given matrix
// n --> order of matrix
// sum --> given sum for which we need to find pair
static void pairSum(int mat[][], int n, int sum) {

    // First sort all the rows in ascending order
    for (int i = 0; i < n; i++)
    Arrays.sort(mat[i]);

    // Select i'th row and find pair for element in i'th
    // row in j'th row whose summation is equal to given sum
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            int left = 0, right = n - 1;
            while (left < n && right >= 0) {
                if ((mat[i][left] + mat[j][right]) == sum) {
                System.out.print("(" + mat[i][left] + ", " +
                                     mat[j][right] + "), ");
                left++;
```

```
                right--;
                }
                else {
                    if ((mat[i][left] + mat[j][right]) < sum)
                        left++;
                    else
                        right--;
                }
            }
        }
    }
}

// Driver code
public static void main(String[] args) {
    int n = 4, sum = 11;
    int mat[]
        [] = {{1 ,  3,  2,  4},
              {5 ,  8,  7,  6},
              {9 , 10, 13, 11},
              {12,  0, 14, 15}};
    pairSum(mat, n, sum);
}
}
// This code is contributed by Anant Agarwal.
```

Output:

```
(1, 10), (3, 8), (2, 9), (4, 7), (11, 0)
```

Time complexity : O(n^2logn + n^3)
Auxiliary space : O(1)

Method 3 (Hashing)

1. Create an empty hash table and store all elements of matrix in hash as key and their locations as values.
2. Traverse the matrix again to check for every element whether its pair exists in hash table or not. If exists, then compare row numbers. If row numbers are not same, then print the pair.

CPP

```
// C++ program to find pairs with given sum such
```

```
// the two elements of pairs are from different rows
#include<bits/stdc++.h>
using namespace std;

const int MAX = 100;

// Function to find pair for given sum in matrix
// mat[][] --> given matrix
// n --> order of matrix
// sum --> given sum for which we need to find pair
void pairSum(int mat[][MAX], int n, int sum)
{
    // Create a hash and store all elements of matrix
    // as keys, and row and column numbers as values
    unordered_map<int, pair<int, int> > hm;
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            hm[mat[i][j]] = make_pair(i, j);

    // Traverse the matrix again to check for every
    // element whether its pair exists or not.
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            // Look for remaining sum in hash
            int remSum = sum - mat[i][j];
            auto it = hm.find(remSum); // it is an iterator
                                       // of unordered_map type

            // If an element with value equal to remaining sum exists
            if (it != hm.end())
            {
                // Find row and column numbers of element with
                // value equal to remaining sum.
                pair<int, int> p = it->second;
                int row = p.first, col = p.second;

                // If row number of pair is not same as current
                // row, then print it as part of result.
                // Second condition is there to make sure that a
                // pair is printed only once.
                if (row != i && row > i)
                   cout << "(" << mat[i][j] << ", "
                        << mat[row][col] << "), ";
            }
        }
    }
```

```
}

// Driver program
int main()
{
    int n = 4, sum = 11;
    int mat[][MAX]= {{1, 3, 2, 4},
                     {5, 8, 7, 6},
                     {9, 10, 13, 11},
                     {12, 0, 14, 15}};
    pairSum(mat, n, sum);
    return 0;
}
```

Output :

```
(1, 10), (3, 8), (2, 9), (4, 7), (11, 0),
```

One important thing is, when we traverse a matrix, a pair may be printed twice. To make sure that a pair is printed only once, we check if row number of other element picked from hash table is more than row number of current element.

Time Complexity : $O(n^2)$ under the assumption that hash table insert and search operations take O(1) time.

Source

https://www.geeksforgeeks.org/find-pairs-given-sum-elements-pair-different-rows/

Chapter 108

Find smallest range containing elements from k lists

Find smallest range containing elements from k lists - GeeksforGeeks

Given k sorted lists of integers of size n each, find the smallest range that includes at least element from each of the k lists. If more than one smallest ranges are found, print any one of them.

Example :

```
Input:
K = 3
arr1[] : [4, 7, 9, 12, 15]
arr2[] : [0, 8, 10, 14, 20]
arr3[] : [6, 12, 16, 30, 50]
Output:
The smallest range is [6 8]
Explanation: Smallest range is formed by
number 7 from first list, 8 from second
list and 6 from third list.
```

```
Input:
k = 3
arr1[] : [4, 7]
arr2[] : [1, 2]
arr3[] : [20, 40]
The smallest range is [2 20]
```

Naive approach : The idea is to maintain pointers to every list using array ptr[k].Below are the steps :

1. Initially the index of every list is 0,therefore initialize every element of ptr[0..k] to 0;
2. Repeat the following steps until atleast one list exhausts :
    - Now find the minimum and maximum value among the current elements of all the list pointed by the ptr[0...k] array.
    - Now update the minrange if current (max-min) is less than minrange.
    - increment the pointer pointing to current minimum element.

```
 // C++ program to finds out smallest range that includes
// elements from each of the given sorted lists.
#include <bits/stdc++.h>

using namespace std;

#define N 5

// array for storing the current index of list i
int ptr[501];

// This function takes an k sorted lists in the form of
// 2D array as an argument. It finds out smallest range
// that includes elements from each of the k lists.
void findSmallestRange(int arr[][N], int n, int k)
{
      int i,minval,maxval,minrange,minel,maxel,flag,minind;

      //initializing to 0 index;
      for(i = 0;i <= k;i++)
        ptr[i] = 0;

      minrange = INT_MAX;

      while(1)
      {
          // for mainting the index of list containing the minimum element
          minind = -1;
          minval = INT_MAX;
          maxval = INT_MIN;
          flag = 0;

          //iterating over all the list
          for(i = 0;i < k;i++)
          {
              // if every element of list[i] is traversed then break the loop
              if(ptr[i] == n)
              {
                flag = 1;
                break;
              }
```

```
            // find minimum value among all the list elements pointing by the ptr[] array
            if(ptr[i] < n && arr[i][ptr[i]] < minval)
            {
                minind=i;  // update the index of the list
                minval=arr[i][ptr[i]];
            }
            // find maximum value among all the list elements pointing by the ptr[] array
            if(ptr[i] < n && arr[i][ptr[i]] > maxval)
            {
                maxval = arr[i][ptr[i]];
            }
        }

        //if any list exhaust we will not get any better answer ,so break the while loop
        if(flag)
          break;

        ptr[minind]++;

        //updating the minrange
        if((maxval-minval) < minrange)
        {
            minel = minval;
            maxel = maxval;
            minrange = maxel - minel;
        }
    }

    printf("The smallest range is [%d , %d]\n",minel,maxel);
}

// Driver program to test above function
int main()
{
    int arr[][N] = {
                    {4, 7, 9, 12, 15},
                    {0, 8, 10, 14, 20},
                    {6, 12, 16, 30, 50}
                    };

    int k = sizeof(arr)/sizeof(arr[0]);

    findSmallestRange(arr,N,k);

    return 0;
}
// This code is contributed by Aditya Krishna Namdeo
```

Time complexity : $O(n^2 k)$

A Better efficient approach is to use min heap. Below are the steps –

1. Create a min heap of size k and insert first elements of all k lists into the heap.
2. Maintain two variables min and max to store minimum and maximum values present in the heap at any point. Note min will always contain value of the root of the heap.
3. Repeat following steps
    - Get minimum element from heap (minimum is always at root) and compute the range.
    - Replace heap root with next element of the list from which the min element is extracted. After replacing the root, heapify the tree. Update max if next element is greater. If the list doesn't have any more elements, break the loop.

Below is C++ implementation of above approach –

```
 // C++ program to finds out smallest range that includes
// elements from each of the given sorted lists.
#include<iostream>
#include<limits.h>
using namespace std;

#define N 5

// A min heap node
struct MinHeapNode
{
    int element; // The element to be stored
    int i; // index of the list from which the element is taken
    int j; // index of the next element to be picked from list
};

// Prototype of a utility function to swap two min heap nodes
void swap(MinHeapNode *x, MinHeapNode *y);

// A class for Min Heap
class MinHeap
{
    MinHeapNode *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor: creates a min heap of given size
    MinHeap(MinHeapNode a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int );
```

```
    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }

    // to get the root
    MinHeapNode getMin() { return harr[0]; }

    // to replace root with new node x and heapify() new root
    void replaceMin(MinHeapNode x) { harr[0] = x; MinHeapify(0); }
};

// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(MinHeapNode a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// A recursive method to heapify a subtree with root at
// given index. This method assumes that the subtrees
// are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size &&
        harr[l].element < harr[i].element)
        smallest = l;
    if (r < heap_size &&
        harr[r].element < harr[smallest].element)
        smallest = r;
    if (smallest != i)
    {
        swap(harr[i], harr[smallest]);
        MinHeapify(smallest);
    }
}

// This function takes an k sorted lists in the form of
```

```
// 2D array as an argument. It finds out smallest range
// that includes elements from each of the k lists.
void findSmallestRange(int arr[][N], int k)
{
    // Create a min heap with k heap nodes. Every heap node
    // has first element of an list
    int range = INT_MAX;
    int min = INT_MAX, max = INT_MIN;
    int start, end;

    MinHeapNode *harr = new MinHeapNode[k];
    for (int i = 0; i < k; i++)
    {
        harr[i].element = arr[i][0]; // Store the first element
        harr[i].i = i; // index of list
        harr[i].j = 1; // Index of next element to be stored
                       // from list

        // store max element
        if (harr[i].element > max)
            max = harr[i].element;
    }

    MinHeap hp(harr, k); // Create the heap

    // Now one by one get the minimum element from min
    // heap and replace it with next element of its list
    while (1)
    {
        // Get the minimum element and store it in output
        MinHeapNode root = hp.getMin();

        // update min
        min = hp.getMin().element;

        // update range
        if (range > max - min + 1)
        {
            range = max - min + 1;
            start = min;
            end = max;
        }

        // Find the next element that will replace current
        // root of heap. The next element belongs to same
        // list as the current root.
        if (root.j < N)
        {
```

```
            root.element = arr[root.i][root.j];
            root.j += 1;

            // update max element
            if (root.element > max)
                max = root.element;
        }

        // break if we have reached end of any list
        else break;

        // Replace root with next element of list
        hp.replaceMin(root);
    }

    cout << "The smallest range is " << "["
         << start << " " << end << "]" << endl;;
}

// Driver program to test above functions
int main()
{
    int arr[][N] = {
                    {4, 7, 9, 12, 15},
                    {0, 8, 10, 14, 20},
                    {6, 12, 16, 30, 50}
                    };
    int k = sizeof(arr)/sizeof(arr[0]);

    findSmallestRange(arr, k);

    return 0;
}
```

Output :

```
The smallest range is [6 8]
```

Time Complexity: The while loop inside findSmallestRange() function can run maximum n*k times. In every iteration of loop, we call heapify which takes O(Logk) time. Therefore, the time complexity is O(nk Logk).

Source

https://www.geeksforgeeks.org/find-smallest-range-containing-elements-from-k-lists/

Chapter 109

Find subarray with given sum | Set 2 (Handles Negative Numbers)

Find subarray with given sum | Set 2 (Handles Negative Numbers) - GeeksforGeeks

Given an unsorted array of integers, find a subarray which adds to a given number. If there are more than one subarrays with sum as the given number, print any of them.

Examples:

```
Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33
Ouptut: Sum found between indexes 2 and 4

Input: arr[] = {10, 2, -2, -20, 10}, sum = -10
Ouptut: Sum found between indexes 0 to 3

Input: arr[] = {-10, 0, 2, -2, -20, 10}, sum = 20
Ouptut: No subarray with given sum exists
```

We have discussed a solution that do not handles negative integers here. In this post, negative integers are also handled.

A simple solution is to consider all subarrays one by one and check if sum of every subarray is equal to given sum or not. The complexity of this solution would be O(n^2).

An efficient way is to use a map. The idea is to maintain sum of elements encountered so far in a variable (say curr_sum). Let the given number is sum. Now for each element, we check if curr_sum – sum exists in the map or not. If we found it in the map that means, we have a subarray present with given sum, else we insert curr_sum into the map and proceed to next element. If all elements of the array are processed and we didn't find any subarray with given sum, then subarray doesn't exists.

Below is C++ implementation of above idea –

```
 // C++ program to print subarray with sum as given sum
#include<bits/stdc++.h>
using namespace std;

// Function to print subarray with sum as given sum
void subArraySum(int arr[], int n, int sum)
{
    // create an empty map
    unordered_map<int, int> map;

    // Maintains sum of elements so far
    int curr_sum = 0;

    for (int i = 0; i < n; i++)
    {
        // add current element to curr_sum
        curr_sum = curr_sum + arr[i];

        // if curr_sum is equal to target sum
        // we found a subarray starting from index 0
        // and ending at index i
        if (curr_sum == sum)
        {
            cout << "Sum found between indexes "
                 << 0 << " to " << i << endl;
            return;
        }

        // If curr_sum - sum already exists in map
        // we have found a subarray with target sum
        if (map.find(curr_sum - sum) != map.end())
        {
            cout << "Sum found between indexes "
                 << map[curr_sum - sum] + 1
                 << " to " << i << endl;
            return;
        }

        map[curr_sum] = i;
    }

    // If we reach here, then no subarray exists
    cout << "No subarray with given sum exists";
}

// Driver program to test above function
```

```
int main()
{
    int arr[] = {10, 2, -2, -20, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = -10;

    subArraySum(arr, n, sum);

    return 0;
}
```

Output:

```
Sum found between indexes 0 to 3
```

Time complexity of above solution is O(n) as we are doing only one traversal of the array.

Auxiliary space used by the program is O(n).

Source

https://www.geeksforgeeks.org/find-subarray-with-given-sum-in-array-of-integers/

Chapter 110

Find substrings that contain all vowels

Find substrings that contain all vowels - GeeksforGeeks

We have been given a string in lowercase alphabets. We need to print substrings that contain all the vowels at-least one time and there are no consonants (non-vowel characters) present in the substrings.

Examples:

```
Input : str = aeoibddaeoiud
Output : aeoiu

Input : str = aeoibsddaeiouudb
Output : aeiou, aeiouu
```

Reference :- Samsung Interview Questions.

We use a hashing based technique and start traversing the string from the start. For every character, we consider all substrings starting from it. If we encounter a consonant, we move to next starting character. Else, we insert current character in a hash. If all vowels are included, we print current substring.

```
 // CPP program to find all substring that
// contain all vowels
#include <bits/stdc++.h>
using namespace std;

// Returns true if x is vowel.
bool isVowel(char x)
{
```

```
    // Function to check whether a character is
    // vowel or not
    return (x == 'a' || x == 'e' || x == 'i' ||
                        x == 'o' || x == 'u');
}

void FindSubstring(string str)
{
    set<char> hash; // To store vowels

    // Outer loop picks starting character and
    // inner loop picks ending character.
    int n = str.length();
    for (int i = 0; i < n; i++) {
       for (int j = i; j < n; j++) {

            // If current character is not vowel,
            // then no more result substrings
            // possible starting from str[i].
            if (isVowel(str[j]) == false)
              break;

            // If vowel, then we insert it in hash
            hash.insert(str[j]);

            // If all vowels are present in current
            // substring
            if (hash.size() == 5)
                cout << str.substr(i, j-i+1) << " ";
        }

        hash.clear();
    }
}

// Driver code
int main()
{
    string str = "aeoibsddaeiouudb";
    FindSubstring(str);
    return 0;
}
```

Output:

```
aeiou aeiouu
```

Time Complexity : $O(n^2)$

Optimized Solution :
For every character, If current character is vowel then insert into hash. else set flag Start to next substring start from i+1th index. If all vowels are included, we print current substring.

```
 // C++ program to find all substring that
// contain all vowels
#include<bits/stdc++.h>
using namespace std;

// Returns true if x is vowel.
bool isVowel(char x)
{
    // Function to check whether a character is
    // vowel or not
    return (x == 'a' || x == 'e' || x == 'i' ||
                        x == 'o' || x == 'u');
}

// Function to FindSubstrings of string
void FindSubstring(string str)
{
    set<char> hash;  // To store vowels

    int start = 0;
    for (int i=0; i<str.length(); i++)
    {
        // If current character is vowel then
        // insert into hash ,
        if (isVowel(str[i]) == true)
        {
            hash.insert(str[i]);

            // If all vowels are present in current
            // substring
            if (hash.size()==5)
                cout << str.substr(start, i-start+1)
                     << " ";
        }
        else
        {
            start = i+1;
            hash.clear();
        }
    }
}

// Driver Code
```

```
int main()
{
    string str = "aeoibsddaeiouudb";
    FindSubstring(str);
    return 0;
}
```

Output:

```
aeiou aeiouu
```

Thanks to **Kriti Shukla** for suggesting this optimized solution.

Source

https://www.geeksforgeeks.org/find-substrings-contain-vowels/

Chapter 111

Find sum of non-repeating (distinct) elements in an array

Find sum of non-repeating (distinct) elements in an array - GeeksforGeeks

Given an integer array with repeated elements, the task is to find sum of all distinct elements in array.

Examples:

```
Input  : arr[] = {12, 10, 9, 45, 2, 10, 10, 45,10};
Output : 78
Here we take 12, 10, 9, 45, 2 for sum
because it's distinct elements

Input : arr[] = {1, 10, 9, 4, 2, 10, 10, 45 , 4};
Output : 71
```

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one starting from the leftmost element. The inner loop checks if the element is present on left side of it. If present, then ignores the element.

Time Complexity : $O(n^2)$
Auxiliary Space : O(1)

A **Better Solution** of this problem is that using sorting technique we firstly sort all elements of array in ascending order and and find one by one distinct elements in array.

```
 // C++ Find the sum of all non-repeated
// elements in an array
#include<bits/stdc++.h>
using namespace std;
```

```
// Find the sum of all non-repeated elements
// in an array
int findSum(int arr[], int n)
{
    // sort all elements of array
    sort(arr, arr + n);

    int sum = 0;
    for (int i=0; i<n; i++)
    {
        if (arr[i] != arr[i+1])
            sum = sum + arr[i];
    }

    return sum;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 1, 1, 4, 5, 6};
    int n = sizeof(arr)/sizeof(int);
    cout << findSum(arr, n);
    return 0;
}
```

Output:

```
21
```

Time Complexity : O(n log n)
Space Complexity : O(1)

An **Efficient solution** of this problem is that using unordered_set we run a single for loop and which value comes first time its add in sum variable and store in hash table that for next time we not use this value.

```
 // C++ Find the sum of all non- repeated
// elements in an array
#include<bits/stdc++.h>
using namespace std;

// Find the sum of all non-repeated elements
// in an array
void findSum(int arr[],int n)
{
```

```
    int sum = 0;

    // Hash to store all element of array
    unordered_set< int > s;
    for (int i=0; i<n; i++)
    {
        if (s.find(arr[i]) == s.end())
        {
            sum += arr[i];
            s.insert(arr[i]);
        }
    }

    return sum;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 1, 1, 4, 5, 6};
    int n = sizeof(arr)/sizeof(int);
    cout << findSum(arr, n);
    return 0;
}
```

Output:

```
21
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/find-sum-non-repeating-distinct-elements-array/

Chapter 112

Find the Number Occurring Odd Number of Times

Find the Number Occurring Odd Number of Times - GeeksforGeeks

Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times. Find the number in O(n) time & constant space.

Examples :

```
Input : arr = {1, 2, 3, 2, 3, 1, 3}
Output : 3

Input : arr = {5, 7, 2, 7, 5, 2, 5}
Output : 5
```

A **Simple Solution** is to run two nested loops. The outer loop picks all elements one by one and inner loop counts number of occurrences of the element picked by outer loop. Time complexity of this solution is $O(n^2)$.

Below is the implementation of the brute force approach :

C++

```
 // C++ program to find the element
// occurring odd number of times
#include<bits/stdc++.h>
using namespace std;

// Funtion to find the element
// occurring odd number of times
int getOddOccurrence(int arr[], int arr_size)
```

```
{
    for (int i = 0; i < arr_size; i++) {

        int count = 0;

        for (int j = 0; j < arr_size; j++)
        {
            if (arr[i] == arr[j])
                count++;
        }
        if (count % 2 != 0)
            return arr[i];
    }
    return -1;
}

// driver code
int main()
    {
        int arr[] = { 2, 3, 5, 4, 5, 2,
                      4, 3, 5, 2, 4, 4, 2 };
        int n = sizeof(arr) / sizeof(arr[0]);

        // Function calling
        cout << getOddOccurrence(arr, n);

        return 0;
    }
```

Java

```
 // Java program to find the element occurring
// odd number of times
class OddOccurrence {

    // funtion to find the element occurring odd
    // number of times
    static int getOddOccurrence(int arr[], int arr_size)
    {
        int i;
        for (i = 0; i < arr_size; i++) {
            int count = 0;
            for (int j = 0; j < arr_size; j++) {
                if (arr[i] == arr[j])
                    count++;
            }
            if (count % 2 != 0)
                return arr[i];
```

```
        }
        return -1;
    }

    // driver code
    public static void main(String[] args)
    {
        int arr[] = new int[]{ 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };
        int n = arr.length;
        System.out.println(getOddOccurrence(arr, n));
    }
}
// This code has been contributed by Kamal Rawal
```

Python3

```
 # Python program to find the element occurring
# odd number of times

# funtion to find the element occurring odd
# number of times
def getOddOccurrence(arr, arr_size):

    for i in range(0,arr_size):
        count = 0
        for j in range(0, arr_size):
            if arr[i] == arr[j]:
                count+=1

        if (count % 2 != 0):
            return arr[i]

    return -1


# driver code
arr = [2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 ]
n = len(arr)
print(getOddOccurrence(arr, n))

# This code has been contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find the element
// occurring odd number of times
```

```
using System;

class GFG
{
    // Funtion to find the element
    // occurring odd number of times
    static int getOddOccurrence(int []arr, int arr_size)
    {
        for (int i = 0; i < arr_size; i++) {
            int count = 0;

            for (int j = 0; j < arr_size; j++) {
                if (arr[i] == arr[j])
                    count++;
            }
            if (count % 2 != 0)
                return arr[i];
        }
        return -1;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };
        int n = arr.Length;
        Console.Write(getOddOccurrence(arr, n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find the
// element occurring odd
// number of times

// Function to find the element
// occurring odd number of times
function getOddOccurrence(&$arr, $arr_size)
{
    $count = 0;
    for ($i = 0;
         $i < $arr_size; $i++)
    {
```

```
        for ($j = 0;
             $j < $arr_size; $j++)
        {
            if ($arr[$i] == $arr[$j])
                $count++;
        }
        if ($count % 2 != 0)
            return $arr[$i];
    }
    return -1;
}

// Driver code
$arr = array(2, 3, 5, 4, 5, 2,
             4, 3, 5, 2, 4, 4, 2);
$n = sizeof($arr);

// Function calling
echo(getOddOccurrence($arr, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output :

```
5
```

A **Better Solution** is to use Hashing. Use array elements as key and their counts as value. Create an empty hash table. One by one traverse the given array elements and store counts. Time complexity of this solution is O(n). But it requires extra space for hashing.

Program :

Java

```
 //Java program to find the element occurring odd
// number of times
import java.io.*;
import java.util.HashMap;

class OddOccurrence
{
    // funtion to find the element occurring odd
    // number of times
    static int getOddOccurrence(int arr[], int n)
```

```
    {
        HashMap<Integer,Integer> hmap = new HashMap<>();

        // Putting all elements into the HashMap
        for(int i = 0; i < n; i++)
        {
            if(hmap.containsKey(arr[i]))
            {
                int val = hmap.get(arr[i]);

                // If array element is already present then
                // increase the count of that element.
                hmap.put(arr[i], val + 1);
            }
            else

                // if array element is not present then put
                // element into the HashMap and initialize
                // the count to one.
                hmap.put(arr[i], 1);
        }

        // Checking for odd occurrence of each element present
        // in the HashMap
        for(Integer a:hmap.keySet())
        {
            if(hmap.get(a) % 2 != 0)
                return a;
        }
        return -1;
    }

    // driver code
    public static void main(String[] args)
    {
        int arr[] = new int[]{2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
        int n = arr.length;
        System.out.println(getOddOccurrence(arr, n));
    }
}
// This code is contributed by Kamal Rawal
```

Output :

```
5
```

The **Best Solution** is to do bitwise XOR of all the elements. XOR of all elements gives

us odd occurring element. Please note that XOR of two elements is 0 if both elements are same and XOR of a number x with 0 is x.

Below is the implementation of the above approach.

C++

```
 // C++ program to find the element
// occurring odd number of times
#include <bits/stdc++.h>
using namespace std;

// Function to find element occurring
// odd number of times
int getOddOccurrence(int ar[], int ar_size)
{
    int res = 0;
    for (int i = 0; i < ar_size; i++)
        res = res ^ ar[i];

    return res;
}

/* Diver function to test above function */
int main()
{
    int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
    int n = sizeof(ar)/sizeof(ar[0]);

    // Function calling
    cout << getOddOccurrence(ar, n);

    return 0;
}
```

C

```
// C program to find the element
// occurring odd number of times
#include <stdio.h>

// Function to find element occurring
// odd number of times
int getOddOccurrence(int ar[], int ar_size)
{
    int res = 0;
```

```
    for (int i = 0; i < ar_size; i++)
        res = res ^ ar[i];

    return res;
}

/* Diver function to test above function */
int main()
{
    int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
    int n = sizeof(ar) / sizeof(ar[0]);

    // Function calling
    printf("%d", getOddOccurrence(ar, n));
    return 0;
}
```

Java

```
 //Java program to find the element occurring odd number of times

class OddOccurance
{
    int getOddOccurrence(int ar[], int ar_size)
    {
        int i;
        int res = 0;
        for (i = 0; i < ar_size; i++)
        {
            res = res ^ ar[i];
        }
        return res;
    }

    public static void main(String[] args)
    {
        OddOccurance occur = new OddOccurance();
        int ar[] = new int[]{2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
        int n = ar.length;
        System.out.println(occur.getOddOccurrence(ar, n));
    }
}
// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program to find the element occurring odd number of times

def getOddOccurrence(arr):

    # Initialize result
    res = 0

    # Traverse the array
    for element in arr:
        # XOR with the result
        res = res ^ element

    return res

# Test array
arr = [ 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2]

print "%d" % getOddOccurrence(arr)
```

C#

```
 // C# program to find the element
// occurring odd number of times
using System;

class GFG
{
    // Funtion to find the element
    // occurring odd number of times
    static int getOddOccurrence(int []arr, int arr_size)
    {
        int res = 0;
        for (int i = 0; i < arr_size; i++)
        {
            res = res ^ arr[i];
        }
        return res;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };
        int n = arr.Length;
        Console.Write(getOddOccurrence(arr, n));
    }
}
```

```
// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find the
// element occurring odd
// number of times

// Function to find element
// occurring odd number of times
function getOddOccurrence(&$ar, $ar_size)
{
    $res = 0;
    for ($i = 0; $i < $ar_size; $i++)
        $res = $res ^ $ar[$i];

    return $res;
}

// Driver Code
$ar = array(2, 3, 5, 4, 5, 2,
            4, 3, 5, 2, 4, 4, 2);
$n = sizeof($ar);

// Function calling
echo(getOddOccurrence($ar, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output :

```
5
```

Time Complexity: O(n)

Improved By : Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/find-the-number-occurring-odd-number-of-times/

Chapter 113

Find the character in first string that is present at minimum index in second string

Find the character in first string that is present at minimum index in second string - GeeksforGeeks

Given a string **str** and another string **patt**. Find the character in **patt** that is present at the minimum index in **str**. If no character of **patt** is present in **str** then print 'No character present'.

Examples:

```
Input : str = "geeksforgeeks"
        patt = "set"
Output : e
Both e and s of patt are present in str,
but e is present at minimum index, which is 1.

Input : str = "adcffaet"
        patt = "onkl"
Output : No character present
```

Source: OLA Interview Experience | Set 12.

Naive Approach: Using two lopp, find the first index of each character of **patt** in **str**. Print the character having the minimum index. If no character of **patt** is present in **str** then print "No character present".

C++

```
 // C++ implementation to find the character in
// first string that is present at minimum index
// in second string
#include <bits/stdc++.h>
using namespace std;

// function to find the minimum index character
void printMinIndexChar(string str, string patt)
{
    // to store the index of character having
    // minimum index
    int minIndex = INT_MAX;

    // lengths of the two strings
    int m = str.size();
    int n = patt.size();

    // traverse 'patt'
    for (int i = 0; i < n; i++) {

        // for each character of 'patt' traverse 'str'
        for (int j = 0; j < m; j++) {

            // if patt[i] is found in 'str', check if
            // it has the minimum index or not. If yes,
            // then update 'minIndex' and break
            if (patt[i] == str[j] && j < minIndex) {
                minIndex = j;
                break;
            }
        }
    }

    // print the minimum index character
    if (minIndex != INT_MAX)
        cout << "Minimum Index Character = "
             << str[minIndex];

    // if no character of 'patt' is present in 'str'
    else
        cout << "No character present";
}

// Driver program to test above
int main()
{
    string str = "geeksforgeeks";
    string patt = "set";
```

```
    printMinIndexChar(str, patt);
    return 0;
}
```

Java

```
 // Java implementation to find the character in
// first string that is present at minimum index
// in second string

public class GFG
{
    // method to find the minimum index character
    static void printMinIndexChar(String str, String patt)
    {
        // to store the index of character having
        // minimum index
        int minIndex = Integer.MAX_VALUE;

        // lengths of the two strings
        int m = str.length();
        int n = patt.length();

        // traverse 'patt'
        for (int i = 0; i < n; i++) {

            // for each character of 'patt' traverse 'str'
            for (int j = 0; j < m; j++) {

                // if patt.charAt(i)is found in 'str', check if
                // it has the minimum index or not. If yes,
                // then update 'minIndex' and break
                if (patt.charAt(i)== str.charAt(j) && j < minIndex) {
                    minIndex = j;
                    break;
                }
            }
        }

        // print the minimum index character
        if (minIndex != Integer.MAX_VALUE)
            System.out.println("Minimum Index Character = " +
                               str.charAt(minIndex));

        // if no character of 'patt' is present in 'str'
        else
            System.out.println("No character present");
    }
```

```
    // Driver Method
    public static void main(String[] args)
    {
        String str = "geeksforgeeks";
        String patt = "set";
        printMinIndexChar(str, patt);
    }
}
```

C#

```
 // C# implementation to find the character in
// first string that is present at minimum index
// in second string
using System;

class GFG
{
    // method to find the minimum index character
    static void printMinIndexChar(String str, String patt)
    {
        // to store the index of character having
        // minimum index
        int minIndex = int.MaxValue;

        // lengths of the two strings
        int m = str.Length;
        int n = patt.Length;

        // traverse 'patt'
        for (int i = 0; i < n; i++) {

            // for each character of 'patt' traverse 'str'
            for (int j = 0; j < m; j++) {

                // if patt.charAt(i)is found in 'str', check if
                // it has the minimum index or not. If yes,
                // then update 'minIndex' and break
                if (patt[i]== str[j] && j < minIndex) {
                    minIndex = j;
                    break;
                }
            }
        }

        // print the minimum index character
        if (minIndex != int.MaxValue)
```

```
            Console.WriteLine("Minimum Index Character = " +
                                str[minIndex]);

        // if no character of 'patt' is present in 'str'
        else
            Console.WriteLine("No character present");
    }

    // Driver Methoda
    public static void Main()
    {
        String str = "geeksforgeeks";
        String patt = "set";
        printMinIndexChar(str, patt);
    }
}
// This code is contributed by Sam007
```

Output:

```
Minimum Index Character = e
```

Time Complexity: O(mn), where **m** and **n** are the lengths of the two strings.
Auxiliary Space: O(1)

Method 2 Efficient Approach(Hashing): Create a hash table with **(key, value)** tuple represented as **(character, index)** tuple. Store the first index of each character of **str** in the hash table. Now, for each character of **patt** check if it is present in the hash table or not. If present then get its index from the hash table and update **minIndex**(minimum index encountered so far). For no matching character print "No character present". Hash table is implemented using unordered_set in C++.

C++

```
 // C++ implementation to find the character in first
// string that is present at minimum index in second
// string
#include <bits/stdc++.h>
using namespace std;

// function to find the minimum index character
void printMinIndexChar(string str, string patt)
{
    // unordered_map 'um' implemented as hash table
    unordered_map<char, int> um;
```

```
    // to store the index of charcter having
    // minimum index
    int minIndex = INT_MAX;

    // lengths of the two strings
    int m = str.size();
    int n = patt.size();-

    // store the first index of each character of 'str'
    for (int i = 0; i < m; i++)
        if (um.find(str[i]) == um.end())
            um[str[i]] = i;

    // traverse the string 'patt'
    for (int i = 0; i < n; i++)

        // if patt[i] is found in 'um', check if
        // it has the minimum index or not accordingly
        // update 'minIndex'
        if (um.find(patt[i]) != um.end() &&
            um[patt[i]] < minIndex)
            minIndex = um[patt[i]];

    // print the minimum index character
    if (minIndex != INT_MAX)
        cout << "Minimum Index Character = "
             << str[minIndex];

    // if no character of 'patt' is present in 'str'
    else
        cout << "No character present";
}

// Driver program to test above
int main()
{
    string str = "geeksforgeeks";
    string patt = "set";
    printMinIndexChar(str, patt);
    return 0;
}
```

Java

```
 // Java implementation to find the character in
// first string that is present at minimum index
// in second string
```

```
import java.util.HashMap;

public class GFG
{
    // method to find the minimum index character
    static void printMinIndexChar(String str, String patt)
    {
        // map to store the first index of each character of 'str'
        HashMap<Character, Integer> hm = new HashMap<>();

        // to store the index of character having
        // minimum index
        int minIndex = Integer.MAX_VALUE;

        // lengths of the two strings
        int m = str.length();
        int n = patt.length();

        // store the first index of each character of 'str'
        for (int i = 0; i < m; i++)
            if(!hm.containsKey(str.charAt(i)))
                hm.put(str.charAt(i),i);

        // traverse the string 'patt'
        for (int i = 0; i < n; i++)
            // if patt[i] is found in 'um', check if
            // it has the minimum index or not accordingly
            // update 'minIndex'
            if (hm.containsKey(patt.charAt(i)) &&
                hm.get(patt.charAt(i)) < minIndex)
                minIndex = hm.get(patt.charAt(i));

        // print the minimum index character
        if (minIndex != Integer.MAX_VALUE)
            System.out.println("Minimum Index Character = " +
                               str.charAt(minIndex));

        // if no character of 'patt' is present in 'str'
        else
            System.out.println("No character present");
    }

    // Driver Method
    public static void main(String[] args)
    {
        String str = "geeksforgeeks";
        String patt = "set";
        printMinIndexChar(str, patt);
```

```
    }
}
```

Output:

```
Minimum Index Character = e
```

Time Complexity: O(m + n), where **m** and **n** are the lengths of the two strings.
Auxiliary Space: O(d), where **d** is the size of hash table, which is the count of distinct characters in **str**.

Source

https://www.geeksforgeeks.org/find-character-first-string-present-minimum-index-second-string/

Chapter 114

Find the first non-repeating character from a stream of characters

Find the first non-repeating character from a stream of characters - GeeksforGeeks

Given a stream of characters, find the first non-repeating character from stream. You need to tell the first non-repeating character in O(1) time at any moment.

If we follow the first approach discussed here, then we need to store the stream so that we can traverse it one more time to find the first non-repeating character at any moment. If we use extended approach discussed in the same post, we need to go through the count array every time first non-repeating element is queried. We can find the first non-repeating character from stream at any moment without traversing any array.

The idea is to use a DLL (**D**oubly **L**inked **L**ist) to efficiently get the first non-repeating character from a stream. The DLL contains all non-repeating characters in order, i.e., the head of DLL contains first non-repeating character, the second node contains the second non-repeating and so on.
We also maintain two arrays: one array is to maintain characters that are already visited two or more times, we call it repeated[], the other array is array of pointers to linked list nodes, we call it inDLL[]. The size of both arrays is equal to alphabet size which is typically 256.

1. Create an empty DLL. Also create two arrays inDLL[] and repeated[] of size 256. inDLL is an array of pointers to DLL nodes. repeated[] is a boolean array, repeated[x] is true if x is repeated two or more times, otherwise false. inDLL[x] contains pointer to a DLL node if character x is present in DLL, otherwise NULL.
2. Initialize all entries of inDLL[] as NULL and repeated[] as false.
3. To get the first non-repeating character, return character at head of DLL.
4. Following are steps to process a new character 'x' in stream.

- If repeated[x] is true, ignore this character (x is already repeated two or more times in the stream)
- If repeated[x] is false and inDLL[x] is NULL (x is seen first time). Append x to DLL and store address of new DLL node in inDLL[x].
- If repeated[x] is false and inDLL[x] is not NULL (x is seen second time). Get DLL node of x using inDLL[x] and remove the node. Also, mark inDLL[x] as NULL and repeated[x] as true.

Note that appending a new node to DLL is O(1) operation if we maintain tail pointer. Removing a node from DLL is also O(1). So both operations, addition of new character and finding first non-repeating character take O(1) time.

C/C++

```
 // A C++ program to find first non-repeating character
// from a stream of characters
#include <iostream>
#define MAX_CHAR 256
using namespace std;

// A linked list node
struct node
{
    char a;
    struct node *next, *prev;
};

// A utility function to append a character x at the end
// of DLL. Note that the function may change head and tail
// pointers, that is why pointers to these pointers are passed.
void appendNode(struct node **head_ref, struct node **tail_ref,
                char x)
{
    struct node *temp = new node;
    temp->a = x;
    temp->prev = temp->next = NULL;

    if (*head_ref == NULL)
    {
        *head_ref = *tail_ref = temp;
        return;
    }
    (*tail_ref)->next = temp;
    temp->prev = *tail_ref;
    *tail_ref = temp;
}

// A utility function to remove a node 'temp' fromt DLL.
```

```
// Note that the function may change head and tail pointers,
// that is why pointers to these pointers are passed.
void removeNode(struct node **head_ref, struct node **tail_ref,
                struct node *temp)
{
    if (*head_ref == NULL)
        return;

    if (*head_ref == temp)
        *head_ref = (*head_ref)->next;
    if (*tail_ref == temp)
        *tail_ref = (*tail_ref)->prev;
    if (temp->next != NULL)
        temp->next->prev = temp->prev;
    if (temp->prev != NULL)
        temp->prev->next = temp->next;

    delete(temp);
}

void findFirstNonRepeating()
{
    // inDLL[x] contains pointer to a DLL node if x is present
    // in DLL. If x is not present, then inDLL[x] is NULL
    struct node *inDLL[MAX_CHAR];

    // repeated[x] is true if x is repeated two or more times.
    // If x is not seen so far or x is seen only once. then
    // repeated[x] is false
    bool repeated[MAX_CHAR];

    // Initialize the above two arrays
    struct node *head = NULL, *tail = NULL;
    for (int i = 0; i < MAX_CHAR; i++)
    {
        inDLL[i] = NULL;
        repeated[i] = false;
    }

    // Let us consider following stream and see the process
    char stream[] = "geeksforgeeksandgeeksquizfor";
    for (int i = 0; stream[i]; i++)
    {
        char x = stream[i];
        cout << "Reading " << x << " from stream n";

        // We process this character only if it has not occurred
        // or occurred only once. repeated[x] is true if x is
```

```
        // repeated twice or more.s
        if (!repeated[x])
        {
            // If the character is not in DLL, then add this at
            // the end of DLL.
            if (inDLL[x] == NULL)
            {
                appendNode(&head, &tail, stream[i]);
                inDLL[x] = tail;
            }
            else // Otherwise remove this character from DLL
            {
                removeNode(&head, &tail, inDLL[x]);
                inDLL[x] = NULL;
                repeated[x] = true; // Also mark it as repeated
            }
        }

        // Print the current first non-repeating character from
        // stream
        if (head != NULL)
            cout << "First non-repeating character so far is "
                 << head->a << endl;
    }
}

/* Driver program to test above function */
int main()
{
    findFirstNonRepeating();
    return 0;
}
```

Java

```
 //A Java program to find first non-repeating character
//from a stream of characters

import java.util.ArrayList;
import java.util.List;

public class NonReapeatingC
{
    final static int MAX_CHAR = 256;

    static void findFirstNonRepeating()
    {
        // inDLL[x] contains pointer to a DLL node if x is present
```

```
    // in DLL. If x is not present, then inDLL[x] is NULL
    List<Character> inDLL =new ArrayList<Character>();

    // repeated[x] is true if x is repeated two or more times.
    // If x is not seen so far or x is seen only once. then
    // repeated[x] is false
    boolean[] repeated =new boolean[MAX_CHAR];

    // Let us consider following stream and see the process
    String stream = "geeksforgeeksandgeeksquizfor";
    for (int i=0;i < stream.length();i++)
    {
        char x = stream.charAt(i);
        System.out.println("Reading "+ x +" from stream n");

        // We process this character only if it has not occurred
        // or occurred only once. repeated[x] is true if x is
        // repeated twice or more.s
        if(!repeated[x])
        {
            // If the character is not in DLL, then add this at
            // the end of DLL.
            if(!(inDLL.contains(x)))
            {
                inDLL.add(x);
            }
            else    // Otherwise remove this character from DLL
            {
                inDLL.remove((Character)x);
                repeated[x] = true; // Also mark it as repeated
            }
        }

        // Print the current first non-repeating character from
        // stream
        if(inDLL.size() != 0)
        {
            System.out.print("First non-repeating character so far is ");
            System.out.println(inDLL.get(0));
        }
    }
}

/* Driver program to test above function */
public static void main(String[] args)
{
    findFirstNonRepeating();
}
```

```
}
//This code is contributed by Sumit Ghosh
```

Python

```
 # A Python program to find first non-repeating character from
# a stream of characters
MAX_CHAR = 256

def findFirstNonRepeating():

    # inDLL[x] contains pointer to a DLL node if x is present
    # in DLL. If x is not present, then inDLL[x] is NULL
    inDLL = [] * MAX_CHAR

    # repeated[x] is true if x is repeated two or more times.
    #  If x is not seen so far or x is seen only once. then
    # repeated[x] is false
    repeated = [False] * MAX_CHAR

    # Let us consider following stream and see the process
    stream = "geeksforgeeksandgeeksquizfor"
    for i in xrange(len(stream)):
        x = stream[i]
        print "Reading " + x + " from stream"

        # We process this character only if it has not occurred
        # or occurred only once. repeated[x] is true if x is
        # repeated twice or more.s
        if not repeated[ord(x)]:

            # If the character is not in DLL, then add this
            # at the end of DLL
            if not x in inDLL:
                inDLL.append(x)
            else:
                inDLL.remove(x)

        if len(inDLL) != 0:
            print "First non-repeating character so far is ",
            print str(inDLL[0])

# Driver program
findFirstNonRepeating()

# This code is contributed by BHAVYA JAIN
```

Output:

```
Reading g from stream
First non-repeating character so far is g
Reading e from stream
First non-repeating character so far is g
Reading e from stream
First non-repeating character so far is g
Reading k from stream
First non-repeating character so far is g
Reading s from stream
First non-repeating character so far is g
Reading f from stream
First non-repeating character so far is g
Reading o from stream
First non-repeating character so far is g
Reading r from stream
First non-repeating character so far is g
Reading g from stream
First non-repeating character so far is k
Reading e from stream
First non-repeating character so far is k
Reading e from stream
First non-repeating character so far is k
Reading k from stream
First non-repeating character so far is s
Reading s from stream
First non-repeating character so far is f
Reading a from stream
First non-repeating character so far is f
Reading n from stream
First non-repeating character so far is f
Reading d from stream
First non-repeating character so far is f
Reading g from stream
First non-repeating character so far is f
Reading e from stream
First non-repeating character so far is f
Reading e from stream
First non-repeating character so far is f
Reading k from stream
First non-repeating character so far is f
Reading s from stream
First non-repeating character so far is f
Reading q from stream
First non-repeating character so far is f
Reading u from stream
First non-repeating character so far is f
Reading i from stream
First non-repeating character so far is f
```

```
Reading z from stream
First non-repeating character so far is f
Reading f from stream
First non-repeating character so far is o
Reading o from stream
First non-repeating character so far is r
Reading r from stream
First non-repeating character so far is a
```

This article is contributed by **Amit Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/find-first-non-repeating-character-stream-characters/

Chapter 115

Find the first repeated character in a string

Find the first repeated character in a string - GeeksforGeeks

Given a string, find the first repeated character in it. We need to find the character that occurs more than once and whose index of first occurrence is smallest.

Examples:

```
Input:  ch = "geeksforgeeks"
Output: e
e is the first element that repeats

Input:  str = "hello geeks"
Output: l
l is the first element that repeats
```

Simple Solution: The solution is to run two nested loops. Start traversing from left side. For every character, check if it repeats or not. If the character repeats, increment count of

repeating characters. When the count becomes k, return the character. Time Complexity of this solution is $O(n^2)$

We can **Use Sorting** to solve the problem in O(n Log n) time. Following are detailed steps.
1) Copy the given array to an auxiliary array temp[].
2) Sort the temp array using a O(nLogn) time sorting algorithm.
3) Scan the input array from left to right. For every element, count its
occurrences in temp[] using binary search. As soon as we find a characterthat occurs more than once, we return the character.

This step can be done in O(n Log n) time.

An **efficient solution** is to use Hashing to solve this in O(n) time on average.

1. Create an empty hash.
2. Scan each character of input string and insert values to each keys in the hash.
3. When any character appears more than once, hash key value is increment by 1, and return the character.

C/C++

```
 // CPP program to find the first
// repeated character in a string
#include <bits/stdc++.h>
using namespace std;

// Returns first repeating character in str.
char firstRepeating(string &str)
{
    // Creates an empty hashset
    unordered_set<char> h;

    // Traverse the input array from left to right
    for (int i=0; i<str.length(); i++)
    {
        char c = str[i];

        // If element is already in hash set, update x
        // and then break
        if (h.find(c) != h.end())
            return c;

        else // Else add element to hash set
            h.insert(c);
    }

    // If there was no repeated character
    return '\0';
}
```

```
// Driver method to test above method
int main ()
{
    string str = "geeksforgeeks";
    cout << firstRepeating(str);
    return 0;
}
```

Java

```
 // Java program to find the first
// repeated character in a string
import java.util.*;

class Main
{
    // This function prints the first repeated
    // character in str[]
    static char firstRepeating(char str[])
    {
        // Creates an empty hashset
        HashSet<Character> h = new HashSet<>();

        // Traverse the input array from left to right
        for (int i=0; i<=str.length-1; i++)
        {
            char c = str[i];

            // If element is already in hash set, update x
            // and then break
            if (h.contains(c))
                return c;

            else // Else add element to hash set
                h.add(c);
        }

        return '\0';
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        String str = "geeksforgeeks";
        char[] arr = str.toCharArray();
        System.out.println(firstRepeating(arr));
    }
```

```
}
```

Python

```
 # Python program to find the first
# repeated character in a string
def firstRepeatedChar(str):

    h = {}  # Create empty hash

    # Traverse each characters in string
    # in lower case order
    for ch in str:

        # If character is already present
        # in hash, return char
        if ch in h:
            return ch;

        # Add ch to hash
        else:
            h[ch] = 0

    return '\0'


# Driver code
print(firstRepeatedChar("geeksforgeeks"))
```

Output:

```
e
```

Further Optimization : Size of alphabet is typically small and fixed. In case of ASCII, size is 256. So we can create a fixed size array and use character ASCII values as index.

Similar Problem: finding first non-repeated character in a string.

Source

https://www.geeksforgeeks.org/find-the-first-repeated-character-in-a-string/

Chapter 116

Find the first repeating element in an array of integers

Find the first repeating element in an array of integers - GeeksforGeeks

Given an array of integers, find the first repeating element in it. We need to find the element that occurs more than once and whose index of first occurrence is smallest.

Examples:

```
Input:  arr[] = {10, 5, 3, 4, 3, 5, 6}
Output: 5 [5 is the first element that repeats]

Input:  arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10}
Output: 6 [6 is the first element that repeats]
```

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one, the inner loop checks whether the element is repeated or not. Once we find an element that repeats, we break the loops and print the element. Time Complexity of this solution is $O(n^2)$

We can **Use Sorting** to solve the problem in O(nLogn) time. Following are detailed steps.
1) Copy the given array to an auxiliary array temp[].
2) Sort the temp array using a O(nLogn) time sorting algorithm.
3) Scan the input array from left to right. For every element, count its occurrences in temp[] using binary search. As soon as we find an element that occurs more than once, we return the element. This step can be done in O(nLogn) time.

We can **Use Hashing** to solve this in O(n) time on average. The idea is to traverse the given array from right to left and update the minimum index whenever we find an element that has been visited on right side. Thanks to Mohammad Shahid for suggesting this solution.

Following are C++ and Java implementation of this idea.

C++

```
 /* C++ program to find first repeating element in arr[] */
#include<bits/stdc++.h>
using namespace std;

// This function prints the first repeating element in arr[]
void printFirstRepeating(int arr[], int n)
{
    // Initialize index of first repeating element
    int min = -1;

    // Creates an empty hashset
    set<int> myset;

    // Traverse the input array from right to left
    for (int i = n - 1; i >= 0; i--)
    {
        // If element is already in hash set, update min
        if (myset.find(arr[i]) != myset.end())
            min = i;

        else   // Else add element to hash set
            myset.insert(arr[i]);
    }

    // Print the result
    if (min != -1)
        cout << "The first repeating element is " << arr[min];
    else
        cout << "There are no repeating elements";
}

// Driver method to test above method
int main()
{
    int arr[] = {10, 5, 3, 4, 3, 5, 6};

    int n = sizeof(arr) / sizeof(arr[0]);
    printFirstRepeating(arr, n);
}
//This article is contributed by Chhavi
```

Java

```
 /* Java program to find first repeating element in arr[] */
import java.util.*;

class Main
{
```

```
    // This function prints the first repeating element in arr[]
    static void printFirstRepeating(int arr[])
    {
        // Initialize index of first repeating element
        int min = -1;

        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array from right to left
        for (int i=arr.length-1; i>=0; i--)
        {
            // If element is already in hash set, update min
            if (set.contains(arr[i]))
                min = i;

            else   // Else add element to hash set
                set.add(arr[i]);
        }

        // Print the result
        if (min != -1)
          System.out.println("The first repeating element is " + arr[min]);
        else
          System.out.println("There are no repeating elements");
    }

    // Driver method to test above method
    public static void main (String[] args) throws java.lang.Exception
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        printFirstRepeating(arr);
    }
}
```

Output:

```
The first repeating element is 5
```

Source

https://www.geeksforgeeks.org/find-first-repeating-element-array-integers/

Chapter 117

Find the largest area rectangular sub-matrix whose sum is equal to k

Find the largest area rectangular sub-matrix whose sum is equal to k - GeeksforGeeks

Given a 2D matrix **mat**[][] and a value **k**. Find the largest rectangular sub-matrix whose sum is equal to **k**.

Example:

```
Input : mat = { { 1, 7, -6, 5 },
                { -8, 6, 7, -2 },
                { 10, -15, 3, 2 },
                { -5, 2, 0, 9 } }
        k = 7

Output : (Top, Left): (0, 1)
         (Bottom, Right): (2, 3)
         7 -6 5
         6 7 -2
        -15 3 2
```

Naive Approach: Check every possible rectangle in given 2D array having sum equal to 'k' and print the largest one. This solution requires 4 nested loops and time complexity of this solution would be O(n^4).

Efficient Approach: Longest sub-array having sum k for 1-D array can be used to reduce the time complexity to O(n^3). The idea is to fix the left and right columns one by one and find the longest sub-array having sum equal to 'k' for contiguous rows for every left and right column pair. We basically find top and bottom row numbers (which are part of the

largest sub-matrix) for every fixed left and right column pair. To find the top and bottom row numbers, calculate sum of elements in every row from left to right and store these sums in an array say temp[]. So temp[i] indicates sum of elements from left to right in row i. Now, apply Longest sub-array having sum k 1D algorithm on temp[], and get the longest sub-array having sum equal to 'k' of temp[]. This length would be the maximum possible length with left and right as boundary columns. Set the 'top' and 'bottom' row indexes for the left right column pair and calculate the area. In similar manner get the top, bottom, left, right indexes for other sub-matrices having sum equal to 'k' and print the one having maximum area.

```
 // C++ implementation to find the largest area rectangular
// sub-matrix whose sum is equal to k
#include <bits/stdc++.h>
using namespace std;

const int MAX = 100;

// This function basically finds largest 'k'
// sum subarray in arr[0..n-1]. If 'k' sum
// doesn't exist, then it returns false. Else
// it returns true and sets starting and
// ending indexes as start and end.
bool sumEqualToK(int arr[], int& start,
                 int& end, int n, int k)
{
    // unordered_map 'um' implemented
    // as hash table
    unordered_map<int, int> um;
    int sum = 0, maxLen = 0;

    // traverse the given array
    for (int i = 0; i < n; i++) {

        // accumulate sum
        sum += arr[i];

        // when subarray starts from index '0'
        // update maxLength and start and end points
        if (sum == k) {
            maxLen = i + 1;
            start = 0;
            end = i;
        }

        // make an entry for 'sum' if it is
        // not present in 'um'
        if (um.find(sum) == um.end())
            um[sum] = i;
```

```
        // check if 'sum-k' is present in 'um'
        // or not
        if (um.find(sum - k) != um.end()) {

            // update maxLength and start and end points
            if (maxLen < (i - um[sum - k])) {
                maxLen = i - um[sum - k];
                start = um[sum - k] + 1;
                end = i;
            }
        }
    }

    // Return true if maximum length is non-zero
    return (maxLen != 0);
}

// function to find the largest area rectangular
// sub-matrix whose sum is equal to k
void sumZeroMatrix(int mat[][MAX], int row, int col, int k)
{
    // Variables to store the temporary values
    int temp[row], area;
    bool sum;
    int up, down;

    // Variables to store the final output
    int fup = 0, fdown = 0, fleft = 0, fright = 0;
    int maxArea = INT_MIN;

    // Set the left column
    for (int left = 0; left < col; left++) {
        // Initialize all elements of temp as 0
        memset(temp, 0, sizeof(temp));

        // Set the right column for the left column
        // set by outer loop
        for (int right = left; right < col; right++) {
            // Calculate sum between current left
            // and right column for every row 'i'
            for (int i = 0; i < row; i++)
                temp[i] += mat[i][right];

            // Find largest subarray with 'k' sum in
            // temp[]. The sumEqualToK() function also
            // sets values of 'up' and 'down;'. So
            // if 'sum' is true then rectangle exists between
```

```
            // (up, left) and (down, right) which are the
            // boundary values.
            sum = sumEqualToK(temp, up, down, row, k);
            area = (down - up + 1) * (right - left + 1);

            // Compare no. of elements with previous
            // no. of elements in sub-Matrix.
            // If new sub-matrix has more elements
            // then update maxArea and final boundaries
            // like fup, fdown, fleft, fright
            if (sum && maxArea < area) {
                fup = up;
                fdown = down;
                fleft = left;
                fright = right;
                maxArea = area;
            }
        }
    }

    // If there is no change in boundaries
    // than check if mat[0][0] equals 'k'
    // If it is not equal to 'k' then print
    // that no such k-sum sub-matrix exists
    if (fup == 0 && fdown == 0 && fleft == 0 &&
        fright == 0 && mat[0][0] != k) {
        cout << "No sub-matrix with sum " << k << " exists";
        return;
    }

    // Print final values

    cout << "(Top, Left): "
         << "(" << fup << ", " << fleft
         << ")" << endl;

    cout << "(Bottom, Right): "
         << "(" << fdown << ", " << fright
         << ")" << endl;

    for (int j = fup; j <= fdown; j++) {
        for (int i = fleft; i <= fright; i++)
            cout << mat[j][i] << " ";
        cout << endl;
    }
}

// Driver program to test above
```

```
int main()
{
    int mat[][MAX] = { { 1, 7, -6, 5 },
                       { -8, 6, 7, -2 },
                       { 10, -15, 3, 2 },
                       { -5, 2, 0, 9 } };

    int row = 4, col = 4;
    int k = 7;
    sumZeroMatrix(mat, row, col, k);
    return 0;
}
```

Output:

```
(Top, Left): (0, 1)
(Bottom, Right): (2, 3)
7 -6 5
6 7 -2
-15 3 2
```

Time Complexity: O(n^3).
Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/find-the-largest-area-rectangular-sub-matrix-whose-sum-is-equal-to-k/

Chapter 118

Find the length of largest subarray with 0 sum

Find the length of largest subarray with 0 sum - GeeksforGeeks

Given an array of integers, find length of the largest subarray with sum equals to 0.

Examples :

```
Input: arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
Output: 5
The largest subarray with 0 sum is -2, 2, -8, 1, 7

Input: arr[] = {1, 2, 3}
Output: 0
There is no subarray with 0 sum

Input: arr[] = {1, 0, 3}
Output: 1
```

A **simple solution** is to consider all subarrays one by one and check the sum of every subarray. We can run two loops: the outer loop picks a starting point i and the inner loop tries all subarrays starting from i. Time complexity of this method is $O(n^2)$.

Below are implementations of this solution.

C/C++

```
 /* A simple C++ program to find largest subarray with 0 sum */
#include<bits/stdc++.h>
using namespace std;

// Returns length of the largest subarray with 0 sum
```

```
int maxLen(int arr[], int n)
{
    int max_len = 0; // Initialize result

    // Pick a starting point
    for (int i = 0; i < n; i++)
    {
        // Initialize currr_sum for every starting point
        int curr_sum = 0;

        // try all subarrays starting with 'i'
        for (int j = i; j < n; j++)
        {
            curr_sum += arr[j];

            // If curr_sum becomes 0, then update max_len
            // if required
            if (curr_sum == 0)
               max_len = max(max_len, j-i+1);
        }
    }
    return max_len;
}

// Driver program to test above function
int main()
{
    int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Length of the longest 0 sum subarray is "
         << maxLen(arr, n);
    return 0;
}
```

Java

```
 // Java code to find the largest subarray
// with 0 sum
class GFG
{
// Returns length of the largest subarray
// with 0 sum
static int maxLen(int arr[], int n)
{
    int max_len = 0;

    // Pick a starting point
    for (int i = 0; i < n; i++)
```

```
    {
        // Initialize curr_sum for every
        // starting point
        int curr_sum = 0;

        // try all subarrays starting with 'i'
        for (int j = i; j < n; j++)
        {
            curr_sum += arr[j];

            // If curr_sum becomes 0, then update
            // max_len
            if (curr_sum == 0)
                max_len = Math.max(max_len, j-i+1);
        }
    }
    return max_len;
}

public static void main(String args[])
{
    int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
    int n = arr.length;
    System.out.println("Length of the longest 0 sum "+
                       "subarray is "+ maxLen(arr, n));
}
}
// This code is contributed by Kamal Rawal
```

Python

```
 # Python program to find the length of largest subarray with 0 sum

# returns the length
def maxLen(arr):

    # initialize result
    max_len = 0

    # pick a starting point
    for i in range(len(arr)):

        # initialize sum for every starting point
        curr_sum = 0

        # try all subarrays starting with 'i'
        for j in range(i, len(arr)):
```

```
            curr_sum += arr[j]

            # if curr_sum becomes 0, then update max_len
            if curr_sum == 0:
                max_len = max(max_len, j-i+1)

    return max_len


# test array
arr = [15, -2, 2, -8, 1, 7, 10, 13]

print "Length of the longest 0 sum subarray is %d" %  maxLen(arr)
```

C#

```
 // C# code to find the largest
// subarray with 0 sum
using System;

class GFG
{
// Returns length of the
// largest subarray with 0 sum
static int maxLen(int []arr, int n)
{
    int max_len = 0;

    // Pick a starting point
    for (int i = 0; i < n; i++)
    {
        // Initialize curr_sum
        // for every starting point
        int curr_sum = 0;

        // try all subarrays
        // starting with 'i'
        for (int j = i; j < n; j++)
        {
            curr_sum += arr[j];

            // If curr_sum becomes 0,
            // then update max_len
            if (curr_sum == 0)
                max_len = Math.Max(max_len,
                                   j - i + 1);
        }
    }
```

```
    return max_len;
}

// Driver code
static public void Main ()
{
int []arr = {15, -2, 2, -8,
             1, 7, 10, 23};
int n = arr.Length;
Console.WriteLine("Length of the longest 0 sum "+
                  "subarray is "+ maxLen(arr, n));
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// A simple PHP program to find
// largest subarray with 0 sum

// Returns length of the
// largest subarray with 0 sum
function maxLen($arr, $n)
{
    $max_len = 0; // Initialize result

    // Pick a starting point
    for ($i = 0; $i < $n; $i++)
    {
        // Initialize currr_sum
        // for every starting point
        $curr_sum = 0;

        // try all subarrays
        // starting with 'i'
        for ($j = $i; $j < $n; $j++)
        {
            $curr_sum += $arr[$j];

            // If curr_sum becomes 0,
            // then update max_len
            // if required
            if ($curr_sum == 0)
            $max_len = max($max_len,
                           $j - $i + 1);
        }
```

```
    }
    return $max_len;
}

// Driver Code
$arr = array(15, -2, 2, -8,
             1, 7, 10, 23);
$n = sizeof($arr);
echo "Length of the longest 0 " .
             "sum subarray is ",
               maxLen($arr, $n);

// This code is contributed by aj_36
?>
```

Output :

```
Length of the longest 0 sum subarray is 5
```

We can **Use Hashing** to solve this problem in O(n) time. The idea is to iterate through the array and for every element arr[i], calculate sum of elements form 0 to i (this can simply be done as sum += arr[i]). If the current sum has been seen before, then there is a zero sum array. Hashing is used to store the sum values, so that we can quickly store sum and find out whether the current sum is seen before or not.

Following are implementations of the above approach.

C++

```
 // C++ program to find the length of largest subarray
// with 0 sum
#include <bits/stdc++.h>
using namespace std;

// Returns Length of the required subarray
int maxLen(int arr[], int n)
{
    // Map to store the previous sums
    unordered_map<int, int> presum;

    int sum = 0;        // Initialize the sum of elements
    int max_len = 0;    // Initialize result

    // Traverse through the given array
    for(int i=0; i<n; i++)
    {
        // Add current element to sum
        sum += arr[i];
```

```
        if (arr[i]==0 && max_len==0)
            max_len = 1;
        if (sum == 0)
            max_len = i+1;

        // Look for this sum in Hash table
        if(presum.find(sum) != presum.end())
        {
            // If this sum is seen before, then update max_len
            max_len = max(max_len, i-presum[sum]);
        }
        else
        {
            // Else insert this sum with index in hash table
            presum[sum] = i;
        }
    }

    return max_len;
}

// Driver Program to test above function
int main()
{
    int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Length of the longest 0 sum subarray is "
         << maxLen(arr, n);

    return 0;
}
```

Java

```
 // A Java program to find maximum length subarray with 0 sum
import java.util.HashMap;

class MaxLenZeroSumSub {

    // Returns length of the maximum length subarray with 0 sum
    static int maxLen(int arr[])
    {
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        int sum = 0;      // Initialize sum of elements
        int max_len = 0;  // Initialize result
```

```
        // Traverse through the given array
        for (int i = 0; i < arr.length; i++)
        {
            // Add current element to sum
            sum += arr[i];

            if (arr[i] == 0 && max_len == 0)
                max_len = 1;

            if (sum == 0)
                max_len = i+1;

            // Look this sum in hash table
            Integer prev_i = hM.get(sum);

            // If this sum is seen before, then update max_len
            // if required
            if (prev_i != null)
               max_len = Math.max(max_len, i-prev_i);
            else  // Else put this sum in hash table
               hM.put(sum, i);
        }

        return max_len;
    }

    // Drive method
    public static void main(String arg[])
    {
        int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
        System.out.println("Length of the longest 0 sum subarray is "
                           + maxLen(arr));
    }
}
```

Python

```
 # A python program to find maximum length subarray
# with 0 sum in o(n) time

# Returns the maximum length
def maxLen(arr):

    # NOTE: Dictonary in python in implemented as Hash Maps
    # Create an empty hash map (dictionary)
    hash_map = {}
```

```
    # Initialize result
    max_len = 0

    # Initialize sum of elements
    curr_sum = 0

    # Traverse through the given array
    for i in range(len(arr)):

        # Add the current element to the sum
        curr_sum += arr[i]

        if arr[i] is 0 and max_len is 0:
            max_len = 1

        if curr_sum is 0:
            max_len = i+1

        # NOTE: 'in' operation in dictionary to search
        # key takes O(1). Look if current sum is seen
        # before
        if curr_sum in hash_map:
            max_len = max(max_len, i - hash_map[curr_sum] )
        else:

            # else put this sum in dictionary
            hash_map[curr_sum] = i

    return max_len


# test array
arr = [15, -2, 2, -8, 1, 7, 10, 13]

print "Length of the longest 0 sum subarray is %d" %  maxLen(arr)
```

Output :

```
Length of the longest 0 sum subarray is 5
```

Time Complexity of this solution can be considered as O(n) under the assumption that we have good hashing function that allows insertion and retrieval operations in O(1) time.

This article is contributed by **Rahul Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t, Mahin_251

Source

https://www.geeksforgeeks.org/find-the-largest-subarray-with-0-sum/

Chapter 119

Find the longest substring with k unique characters in a given string

Find the longest substring with k unique characters in a given string - GeeksforGeeks

Given a string you need to print longest possible substring that has exactly M unique characters. If there are more than one substring of longest possible length, then print any one of them.

Examples:

```
"aabbcc", k = 1
Max substring can be any one from {"aa" , "bb" , "cc"}.

"aabbcc", k = 2
Max substring can be any one from {"aabb" , "bbcc"}.

"aabbcc", k = 3
There are substrings with exactly 3 unique characters
{"aabbcc" , "abbcc" , "aabbc" , "abbc" }
Max is "aabbcc" with length 6.

"aaabbb", k = 3
There are only two unique characters, thus show error message.
```

Source: Google Interview Question.

Method 1 (Brute Force)
If the length of string is n, then there can be n*(n+1)/2 possible substrings. A simple way is to generate all the substring and check each one whether it has exactly k unique characters

or not. If we apply this brute force, it would take $O(n^2)$ to generate all substrings and O(n) to do a check on each one. Thus overall it would go $O(n^3)$.

We can further improve this solution by creating a hash table and while generating the substrings, check the number of unique characters using that hash table. Thus it would improve up to $O(n^2)$.

Method 2 (Linear Time)
The problem can be solved in O(n). Idea is to maintain a window and add elements to the window till it contains less or equal k, update our result if required while doing so. If unique elements exceeds than required in window, start removing the elements from left side.

Below are C++ and Python implementations of above. The implementations assume that the input string alphabet contains only 26 characters (from 'a' to 'z'). The code can be easily extended to 256 characters.

C++

```
 // C++ program to find the longest substring with k unique
// characters in a given string
#include <iostream>
#include <cstring>
#define MAX_CHARS 26
using namespace std;

// This function calculates number of unique characters
// using a associative array count[]. Returns true if
// no. of characters are less than required else returns
// false.
bool isValid(int count[], int k)
{
    int val = 0;
    for (int i=0; i<MAX_CHARS; i++)
        if (count[i] > 0)
            val++;

    // Return true if k is greater than or equal to val
    return (k >= val);
}

// Finds the maximum substring with exactly k unique chars
void kUniques(string s, int k)
{
    int u = 0; // number of unique characters
    int n = s.length();

    // Associative array to store the count of characters
    int count[MAX_CHARS];
    memset(count, 0, sizeof(count));
```

```
// Traverse the string, Fills the associative array
// count[] and count number of unique characters
for (int i=0; i<n; i++)
{
    if (count[s[i]-'a']==0)
        u++;
    count[s[i]-'a']++;
}

// If there are not enough unique characters, show
// an error message.
if (u < k)
{
    cout << "Not enough unique characters";
    return;
}

// Otherwise take a window with first element in it.
// start and end variables.
int curr_start = 0, curr_end = 0;

// Also initialize values for result longest window
int max_window_size = 1, max_window_start = 0;

// Initialize associative array count[] with zero
memset(count, 0, sizeof(count));

count[s[0]-'a']++;  // put the first character

// Start from the second character and add
// characters in window according to above
// explanation
for (int i=1; i<n; i++)
{
    // Add the character 's[i]' to current window
    count[s[i]-'a']++;
    curr_end++;

    // If there are more than k unique characters in
    // current window, remove from left side
    while (!isValid(count, k))
    {
        count[s[curr_start]-'a']--;
        curr_start++;
    }

    // Update the max window size if required
```

```
        if (curr_end-curr_start+1 > max_window_size)
        {
            max_window_size = curr_end-curr_start+1;
            max_window_start = curr_start;
        }
    }

    cout << "Max sustring is : "
         << s.substr(max_window_start, max_window_size)
         << " with length " << max_window_size << endl;
}

// Driver function
int main()
{
    string s = "aabacbebebe";
    int k = 3;
    kUniques(s, k);
    return 0;
}
```

Python

```
 # Python program to find the longest substring with k unique
# characters in a given string
MAX_CHARS = 26

# This function calculates number of unique characters
# using a associative array count[]. Returns true if
# no. of characters are less than required else returns
# false.
def isValid(count, k):
    val = 0
    for i in xrange(MAX_CHARS):
        if count[i] > 0:
            val += 1

    # Return true if k is greater than or equal to val
    return (k >= val)

# Finds the maximum substring with exactly k unique characters
def kUniques(s, k):
    u = 0    # number of unique characters
    n = len(s)

    # Associative array to store the count
    count = [0] * MAX_CHARS
```

```
# Tranverse the string, fills the associative array
# count[] and count number of unique characters
for i in xrange(n):
    if count[ord(s[i])-ord('a')] == 0:
        u += 1
    count[ord(s[i])-ord('a')] += 1

# If there are not enough unique characters, show
# an error message.
if u < k:
    print "Not enough unique characters"
    return

# Otherwise take a window with first element in it.
# start and end variables.
curr_start = 0
curr_end = 0

# Also initialize values for result longest window
max_window_size = 1
max_window_start = 0

# Initialize associative array count[] with zero
count = [0] * len(count)

count[ord(s[0])-ord('a')] += 1    # put the first character

# Start from the second character and add
# characters in window according to above
# explanation
for i in xrange(1,n):
    # Add the character 's[i]' to current window
    count[ord(s[i])-ord('a')] += 1
    curr_end+=1

    # If there are more than k unique characters in
    # current window, remove from left side
    while not isValid(count, k):
        count[ord(s[curr_start])-ord('a')] -= 1
        curr_start += 1

    # Update the max window size if required
    if curr_end-curr_start+1 > max_window_size:
        max_window_size = curr_end-curr_start+1
        max_window_start = curr_start

print "Max substring is : " + s[max_window_start:] \
        + " with length " + str(max_window_size)
```

```
# Driver function
s = "aabacbebebe"
k = 3
kUniques(s, k)

# This code is contributed by BHAVYA JAIN
```

Output:

```
Max sustring is : cbebebe with length 7
```

Time Complexity: Considering function "isValid()" takes constant time, time complexity of above solution is O(n).

This article is contributed by Gaurav Sharma. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/find-the-longest-substring-with-k-unique-characters-in-a-given-string/

Chapter 120

Find the most frequent digit without using array/string

Find the most frequent digit without using array/string - GeeksforGeeks

Given an integer, find the most occurring digit in it. If two or more digits occur same number of times, then return the highest of them. Input integer is given as an int variable, not as a string or array. Use of hash or array or string is not allowed.

Example:

```
Input:  x = 12234
Output: The most frequent digit is 2

Input:  x = 1223377
Output: The most frequent digit is 7

Input:  x = 5
Output: The most frequent digit is 5

Input:  x = 1000
Output: The most frequent digit is 0
```

We strongly recommend you to minimize your browser and try this yourself first.
We could create a map of size 10 and store count of all digits, but use of any array/string is not allowed.

The idea is simple, we write a function that counts occurrences of a given digit in a given integer. Then we count all digits from 0 to 9 in given integer. We keep updating maximum count whenever count becomes more or same as previous count. Below is C++ implementation.

C

```
 // Finds maximum occurring digit without using any array/string
#include <iostream>
using namespace std;

// Simple function to count occurrences of digit d in x
int countOccurrences(long int x, int d)
{
    int count = 0;  // Initialize count of digit d
    while (x)
    {
        // Increment count if current digit is same as d
        if (x%10 == d)
           count++;
        x = x/10;
    }
    return count;
}

// Returns the max occurring digit in x
int maxOccurring(long int x)
{
   // Handle negative number
   if (x < 0)
      x = -x;

   int result = 0; // Initialize result which is a digit
   int max_count = 1; // Initialize count of result

   // Traverse through all digits
   for (int d=0; d<=9; d++)
   {
      // Count occurrences of current digit
      int count = countOccurrences(x, d);

      // Update max_count and result if needed
      if (count >= max_count)
      {
         max_count = count;
         result = d;
      }
   }
   return result;
}

// Driver program
int main()
```

```
{
    long int x = 1223355;
    cout << "Max occurring digit is " << maxOccurring(x);
    return 0;
}
```

Java

```
 // Finds maximum occurring digit
// without using any array/string
import java.io.*;

class GFG
{

// Simple function to count
// occurrences of digit d in x
static int countOccurrences(int x,
                            int d)
{
    // Initialize count
    // of digit d
    int count = 0;
    while (x > 0)
    {
        // Increment count if
        // current digit is
        // same as d
        if (x % 10 == d)
        count++;
        x = x / 10;
    }
    return count;
}

// Returns the max
// occurring digit in x
static int maxOccurring( int x)
{

// Handle negative number
if (x < 0)
    x = -x;

// Initialize result
// which is a digit
int result = 0;
```

```
// Initialize count
// of result
int max_count = 1;

// Traverse through
// all digits
for (int d = 0; d <= 9; d++)
{
    // Count occurrences
    // of current digit
    int count = countOccurrences(x, d);

    // Update max_count
    // and result if needed
    if (count >= max_count)
    {
        max_count = count;
        result = d;
    }
}
return result;
}

// Driver Code
public static void main (String[] args)
{
    int x = 1223355;
    System.out.println("Max occurring digit is " +
                                maxOccurring(x));

}
}

// This code is contributed
// by akt_mit
```

Output:

```
Max occurring digit is 5
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/find-the-most-frequent-digit-without-using-arraystring/

Chapter 121

Find the only element that appears b times

Find the only element that appears b times - GeeksforGeeks

Given an array where every element occurs a times, except one element which occurs b (a>b) times. Find the element that occurs b times.

Examples:

```
Input : arr[]  = [1, 1, 2, 2, 2, 3, 3, 3]
         a = 3, b = 2
Output : 1
```

Add each number once and multiply the sum by a, we will get a times the sum of each element of the array. Store it as a_sum. Subtract the sum of the whole array from the a_sum and divide the result by (a-b). The number we get is the required number (which appears b times in the array).

```
 // CPP program to find the only element that
// appears b times
#include <bits/stdc++.h>
using namespace std;

int appearsbTimes(int arr[], int n, int a, int b)
{
    unordered_set<int> s;

    int a_sum = 0, sum = 0;

    for (int i = 0; i < n; i++) {
        if (s.find(arr[i]) == s.end()) {
```

```
            s.insert(arr[i]);
            a_sum += arr[i];
        }

        sum += arr[i];
    }

    a_sum = a * a_sum;

    return ((a_sum - sum) / (a - b));
}

int main()
{
    int arr[] = { 1, 1, 2, 2, 2, 3, 3, 3 };
    int a = 3, b = 2;
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << appearsbTimes(arr, n, a, b);
    return 0;
}
```

Output:

```
1
```

Please refer below article for more approaches.

Find the only element that appears k times.

Source

https://www.geeksforgeeks.org/find-element-appears-b-times/

Chapter 122

Find the only repetitive element between 1 to n-1

Find the only repetitive element between 1 to n-1 - GeeksforGeeks

We are given an array arr[] of size n. Numbers are from 1 to (n-1) in random order. The array has only one repetitive element. We need to find the repetitive element.

Examples :

```
Input  : a[] = {1, 3, 2, 3, 4}
Output : 3

Input  : a[] = {1, 5, 1, 2, 3, 4}
Output : 1
```

Method 1 (Simple) We use two nested loops. The outer loop traverses through all elements and the inner loop check if the element picked by outer loop appears anywhere else.

Time Complexity : O(n*n)

Method 2 (Using Sum Formula): We know sum of first n-1 natural numbersis (n – 1)*n/2. We compute sum of array elements and subtract natural number sum from it to find the only missing element.

C++

```
 // CPP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;
```

```
int findRepeating(int arr[], int n)
{
   // Find array sum and subtract sum
   // first n-1 natural numbers from it
   // to find the result.
   return accumulate(arr , arr+n , 0) -
                  ((n - 1) * n/2);
}

// driver code
int main()
{
    int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findRepeating(arr, n);
    return 0;
}
```

Output :

```
8
```

Time Complexity : O(n)
Auxiliary Space : O(1)
Causes overflow for large arrays.

Method 3 (Use Hashing): Use a hash table to store elements visited. If a seen element appears again, we return it.

C++

```
 // CPP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;

int findRepeating(int arr[], int n)
{
   unordered_set<int> s;
   for (int i=0; i<n; i++)
   {
      if (s.find(arr[i]) != s.end())
         return arr[i];
      s.insert(arr[i]);
   }
```

```
   // If input is correct, we should
   // never reach here
   return -1;
}

// driver code
int main()
{
    int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findRepeating(arr, n);
    return 0;
}
```

Output :

```
8
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Method 4(Use XOR): The idea is based on the fact that x ^ x = 0 and x ^ y = y ^ x.
1) Compute XOR of elements from 1 to n-1.
2) Compute XOR of array elements.
3) XOR of above two would be our result.

C++

```
 // CPP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;

int findRepeating(int arr[], int n)
{

   // res is going to store value of
   // 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
   // arr[1] ^ .... arr[n-1]
   int res = 0;
   for (int i=0; i<n-1; i++)
       res = res ^ (i+1) ^ arr[i];
   res = res ^ arr[n-1];
```

```
   return res;
}

// driver code
int main()
{
    int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findRepeating(arr, n);
    return 0;
}
```

Java

```
 // Java program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
class GFG
{

    static int findRepeating(int arr[], int n)
    {

        // res is going to store value of
        // 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
        // arr[1] ^ .... arr[n-1]
        int res = 0;
        for (int i = 0; i < n - 1; i++)
            res = res ^ (i + 1) ^ arr[i];
        res = res ^ arr[n - 1];

        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
        int n = arr.length;
        System.out.println(findRepeating(arr, n));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal.
```

Python3

```
 # Python3 program to find the only
# repeating element in an array where
# elements are from 1 to n-1.

def findRepeating(arr, n):

    # res is going to store value of
    # 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
    # arr[1] ^ .... arr[n-1]
    res = 0
    for i in range(0, n-1):
        res = res ^ (i+1) ^ arr[i]
    res = res ^ arr[n-1]

    return res

# Driver code
arr = [9, 8, 2, 6, 1, 8, 5, 3, 4, 7]
n = len(arr)
print(findRepeating(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find the
// only repeating element
// in an array where elements
// are from 1 to n-1.
using System;

class GFG
{
    static int findRepeating(int []arr,
                             int n)
    {

        // res is going to store
        // value of 1 ^ 2 ^ 3 ..
        // ^ (n-1) ^ arr[0] ^
        // arr[1] ^ .... arr[n-1]
        int res = 0;
        for (int i = 0; i < n - 1; i++)
            res = res ^ (i + 1) ^ arr[i];
        res = res ^ arr[n - 1];

        return res;
    }
```

```
    // Driver code
    public static void Main()
    {
        int []arr = { 9, 8, 2, 6, 1,
                      8, 5, 3, 4, 7 };
        int n = arr.Length;
        Console.Write(findRepeating(arr, n));
    }
}

// This code is contributed
// by Smitha Dinesh Semwal.
```

PHP

```
 <?php
// PHP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.

function findRepeating($arr, $n)
{

    // res is going to store value of
    // 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
    // arr[1] ^ .... arr[n-1]
    $res = 0;
    for($i = 0; $i < $n - 1; $i++)
        $res = $res ^ ($i + 1) ^ $arr[$i];
    $res = $res ^ $arr[$n - 1];

    return $res;
}

    // Driver Code
    $arr =array(9, 8, 2, 6, 1, 8, 5, 3, 4, 7);
    $n = sizeof($arr) ;
    echo findRepeating($arr, $n);

// This code is contributed by ajit
?>
```

Output:

```
8
```

Time Complexity : O(n)
Auxiliary Space : O(1)

Method 5 : Using indexing.
1. Iterate through the array.
2. For every index visit a[index], if it is positive change the sign of element at a[index] index, else print the element.

C++

```
 // CPP program to find the only
// repeating element in an array
// where elements are from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;

// Function to find repeted element
int findRepeating(int arr[], int n)
{
    int missingElement = 0;

    // indexing based
    for (int i = 0; i < n; i++){

        int element = arr[abs(arr[i])];

        if(element < 0){
            missingElement = arr[i];
            break;
        }

    arr[abs(arr[i])] = -arr[abs(arr[i])];
}

return abs(missingElement);

}

// driver code
int main()
{
    int arr[] = { 5, 4, 3, 9, 8,
                  9, 1, 6, 2, 5};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findRepeating(arr, n);
```

```
    return 0;
}
```

Java

```
 // Java program to find the only
// repeating element in an array
// where elements are from 1 to n-1.
import java.lang.Math.*;

class GFG
{
    // Function to find repeted element
    static int findRepeating(int arr[], int n)
    {
        int missingElement = 0;

        // indexing based
        for (int i = 0; i < n; i++){

            int element = arr[Math.abs(arr[i])];

            if(element < 0){
                missingElement = arr[i];
                break;
            }

        arr[Math.abs(arr[i])] = -arr[Math.abs(arr[i])];
    }

    return Math.abs(missingElement);

    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 5, 4, 3, 9, 8,
                      9, 1, 6, 2, 5};

        int n = arr.length;

        System.out.println(findRepeating(arr, n));

    }
}
// This code is contributed by
// Smitha Dinesh Semwal.
```

Python3

```
 # Python3 program to find the only
# repeating element in an array
# where elements are from 1 to n-1.

# Function to find repeted element
def findRepeating(arr, n):

    missingElement = 0

    # indexing based
    for i in range(0, n):

        element = arr[abs(arr[i])]

        if(element < 0):
            missingElement = arr[i]
            break

        arr[abs(arr[i])] = -arr[abs(arr[i])]

    return abs(missingElement)

# Driver code
arr = [5, 4, 3, 9, 8, 9, 1, 6, 2, 5]
n = len(arr)
print(findRepeating(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

PHP

```
 <?php
// PHP program to find the only
// repeating element in an array
// where elements are from 1 to n-1.

// Function to find repeted element
function findRepeating($arr, $n)
{
    $missingElement = 0;

    // indexing based
    for ($i = 0; $i < $n; $i++)
    {
```

```
        $element = $arr[abs($arr[$i])];

        if($element < 0)
        {
            $missingElement = $arr[$i];
            break;
        }

    $arr[abs($arr[$i])] = -$arr[abs($arr[$i])];
}

return abs($missingElement);

}

// Driver Code
$arr = array (5, 4, 3, 9, 8,
              9, 1, 6, 2, 5);

$n = sizeof($arr);

echo findRepeating($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
9
```

Time Complexiy : O(n)
Auxiliary Space : O(1)

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-repetitive-element-1-n-1/

Chapter 123

Find the overlapping sum of two arrays

Find the overlapping sum of two arrays - GeeksforGeeks

Given two arrays A[] and B[] having n unique elements each. The task is to find the overlapping sum of the two arrays. That is the sum of elements which is common in both of the arrays.

Note: Elements in the arrays are unique. That is the array does not contains duplicates.

Examples:

```
Input : A[] = {1, 5, 3, 8}
        B[] = {5, 4, 6, 7}
Output : 10
Explanation : The element which is common in both arrays is 5.
Therefore, the overlapping sum will be (5+5) = 10

Input : A[] = {1, 5, 3, 8}
        B[] = {5, 1, 8, 3}
Output : 99
```

Brute Force Method : The simple approach is that for each element in A[] check whether it is present in B[] and if it is present in B[] then add that number two times(once for A[] and once for B[]) to the sum. Repeat this procedure for all elements in the array A[].
Time Complexity: O(n^2).

Efficient Method : An efficient method is to use Hashing. Traverse both of the arrays and insert the elements into a hash table to keep track of the count of elements. Add the elements to sum whose count equals to two.

Below is the implementation of above approach:

```
 // CPP program to find overlapping sum
#include <bits/stdc++.h>
using namespace std;

// Function for calculating
// overlapping sum of two array
int findSum(int A[], int B[], int n)
{
    // unordered map to store count of
    // elements
    unordered_map<int,int> hash;

    // insert elements of A[] into
    // unordered_map
    for(int i=0;i<n;i++)
    {
        if(hash.find(A[i])==hash.end())
        {
            hash.insert(make_pair(A[i],1));
        }
        else
        {
            hash[A[i]]++;
        }
    }

    // insert elements of B[] into
    // unordered_map
    for(int i=0;i<n;i++)
    {
        if(hash.find(B[i])==hash.end())
        {
            hash.insert(make_pair(B[i],1));
        }
        else
        {
            hash[B[i]]++;
        }
    }

    // calculate overlapped sum
    int sum = 0;
    for(auto itr = hash.begin(); itr!=hash.end(); itr++)
    {
        if((itr->second)==2)
        {
            sum += (itr->first)*2;
        }
```

```
    }

    return sum;
}

// driver code
int main()
{
    int A[] = { 5, 4, 9, 2, 3 };
    int B[] = { 2, 8, 7, 6, 3 };

    // size of array
    int n = sizeof(A) / sizeof(A[0]);

    // function call
    cout << findSum(A, B, n);

    return 0;
}
```

Output:

```
52
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Source

https://www.geeksforgeeks.org/find-the-overlapping-sum-of-two-arrays/

Chapter 124

Find the smallest window in a string containing all characters of another string

Find the smallest window in a string containing all characters of another string - GeeksforGeeks

Given two strings string1 and string2, find the smallest substring in string1 containing all characters of string2 efficiently.
For Example:

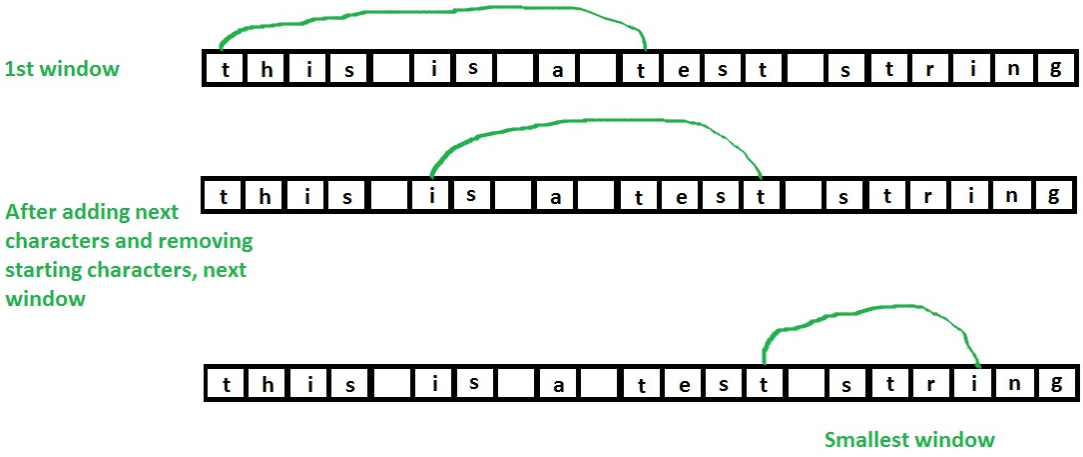
```
Input :  string = "this is a test string"
         pattern = "tist"
Output :  Minimum window is "t stri"
Explanation: "t stri" contains all the characters
              of pattern.

Input :  string = "geeksforgeeks"
         pattern = "ork"
Output :  Minimum window is "ksfor"
```

Method 1 (Brute force solution)
1- Generate all substrings of string1 (“this is a test string”)
2- For each substring, check whether the substring contains all characters of string2 (“tist”)
3- Finally, print the smallest substring containing all characters of string2.

Method 2 (Efficient Solution)

```
1- First check if length of string is less than
   the length of given pattern, if yes
       then "no such window can exist ".
2- Store the occurrence of characters of given
   pattern in a hash_pat[].
3- Start matching the characters of pattern with
   the characters of string i.e. increment count
   if a character matches
4- Check if (count == length of pattern ) this
   means a window is found
5- If such window found, try to minimize it by
   removing extra characters from beginning of
   current window.
6- Update min_length.
7- Print the minimum length window.
```

Diagram to explain above algorithm:



Below is program to implement above algorithm

C++

```
 // C++ program to find smallest window containing
// all characters of a pattern.
#include<bits/stdc++.h>
using namespace std;

const int no_of_chars = 256;

// Function to find smallest window containing
// all characters of 'pat'
string findSubString(string str, string pat)
{
    int len1 = str.length();
```

```
int len2 = pat.length();

// check if string's length is less than pattern's
// length. If yes then no such window can exist
if (len1 < len2)
{
    cout << "No such window exists";
    return "";
}

int hash_pat[no_of_chars] = {0};
int hash_str[no_of_chars] = {0};

// store occurrence ofs characters of pattern
for (int i = 0; i < len2; i++)
    hash_pat[pat[i]]++;

int start = 0, start_index = -1, min_len = INT_MAX;

// start traversing the string
int count = 0; // count of characters
for (int j = 0; j < len1 ; j++)
{
    // count occurrence of characters of string
    hash_str[str[j]]++;

    // If string's char matches with pattern's char
    // then increment count
    if (hash_pat[str[j]] != 0 &&
        hash_str[str[j]] <= hash_pat[str[j]] )
        count++;

    // if all the characters are matched
    if (count == len2)
    {
        // Try to minimize the window i.e., check if
        // any character is occurring more no. of times
        // than its occurrence in pattern, if yes
        // then remove it from starting and also remove
        // the useless characters.
        while ( hash_str[str[start]] > hash_pat[str[start]]
            || hash_pat[str[start]] == 0)
        {

            if (hash_str[str[start]] > hash_pat[str[start]])
                hash_str[str[start]]--;
            start++;
        }
```

```
            // update window size
            int len_window = j - start + 1;
            if (min_len > len_window)
            {
                min_len = len_window;
                start_index = start;
            }
        }
    }

    // If no window found
    if (start_index == -1)
    {
    cout << "No such window exists";
    return "";
    }

    // Return substring starting from start_index
    // and length min_len
    return str.substr(start_index, min_len);
}

// Driver code
int main()
{
    string str = "this is a test string";
    string pat = "tist";

    cout << "Smallest window is : n"
        << findSubString(str, pat);
    return 0;
}
```

Java

```
 // Java program to find smallest window containing
// all characters of a pattern.

public class GFG
{
    static final int no_of_chars = 256;

    // Function to find smallest window containing
    // all characters of 'pat'
    static String findSubString(String str, String pat)
    {
        int len1 = str.length();
```

```
int len2 = pat.length();

// check if string's length is less than pattern's
// length. If yes then no such window can exist
if (len1 < len2)
{
    System.out.println("No such window exists");
    return "";
}

int hash_pat[] = new int[no_of_chars];
int hash_str[] = new int[no_of_chars];

// store occurrence ofs characters of pattern
for (int i = 0; i < len2; i++)
    hash_pat[pat.charAt(i)]++;

int start = 0, start_index = -1, min_len = Integer.MAX_VALUE;

// start traversing the string
int count = 0;  // count of characters
for (int j = 0; j < len1 ; j++)
{
    // count occurrence of characters of string
    hash_str[str.charAt(j)]++;

    // If string's char matches with pattern's char
    // then increment count
    if (hash_pat[str.charAt(j)] != 0 &&
        hash_str[str.charAt(j)] <= hash_pat[str.charAt(j)] )
        count++;

    // if all the characters are matched
    if (count == len2)
    {
        // Try to minimize the window i.e., check if
        // any character is occurring more no. of times
        // than its occurrence  in pattern, if yes
        // then remove it from starting and also remove
        // the useless characters.
        while ( hash_str[str.charAt(start)] > hash_pat[str.charAt(start)]
               || hash_pat[str.charAt(start)] == 0)
        {

            if (hash_str[str.charAt(start)] > hash_pat[str.charAt(start)])
                hash_str[str.charAt(start)]--;
            start++;
        }
```

```
                // update window size
                int len_window = j - start + 1;
                if (min_len > len_window)
                {
                    min_len = len_window;
                    start_index = start;
                }
            }
        }

        // If no window found
        if (start_index == -1)
        {
           System.out.println("No such window exists");
           return "";
        }

        // Return substring starting from start_index
        // and length min_len
        return str.substring(start_index, start_index + min_len);
    }

    // Driver Method
    public static void main(String[] args)
    {
        String str = "this is a test string";
        String pat = "tist";

       System.out.print("Smallest window is :  n" +
                        findSubString(str, pat));
    }
}
```

Output:

```
Smallest window is :
t stri
```

Improved By : kamikaze101

Source

https://www.geeksforgeeks.org/find-the-smallest-window-in-a-string-containing-all-characters-of-another-string/

Chapter 125

Find the starting indices of the substrings in string (S) which is made by concatenating all words from a list(L)

Find the starting indices of the substrings in string (S) which is made by concatenating all words from a list(L) - GeeksforGeeks

You are given a string S, and a list of words L i.e array/vector of strings (Words in list L are all of the same length). Find the starting indices of the substrings in string S, which contains all the words present in list L.

The order of words of list L appearing inside string S does not matter i.e if string S is "barfooapplefoobar" and list of words (L) is ["foo", "bar"] then we have to look for substrings "foobar", "barfoo" in string S.

Note : **Words inside the list L can repeat.**

Examples :

```
Input : S: "barfoothefoobarman"
        L: ["foo", "bar"]
Output : 0 9
Explanation :
// at index 0 : barfoo
// at index 9 : foobar

Input : S: "catbatatecatatebat"
        L: ["cat", "ate", "bat"]
Output : 0 3 9
```

```
Explanation :
// at index 0 : catbatate
// at index 3 : batatecat
// at index 9 : catatebat

Input : S : "abcdababcd"
        L : ["ab", "ab", "cd"]
Output : 0 2 4
Explanation :
// at index 0 : abcdab
// at index 2 : cdabab
// at index 4 : ababcd

Input : S : "abcdababcd"
        L : ["ab", "ab"]
Output : 4
```

Approach :
We can use Hashing Technique to solve the above problem. Let's see the steps :

1. Declare a map (**hash_map**) which stores all words of List L corresponding to their occurrences inside list L.
2. Traverse through all possible substrings in string S which are equal to size_L(total number of characters produced if all the words in list L are concatenated).
3. Create a temporary map (**temp_hash_map**) and initialize it with original map(**hash_map**) for every possible substring.
4. Extract the words from the substring and if the word is present in temp_hash_map we decrease it's corresponding count, if it's not present in temp_hash_map we simply break.
5. After traversing the substring we traverse temp_hash_map and look for any key which has it's count > 0. If we found no such key it means that all the words in list L were found in substring and store the given starting index of the substring, if we find a key which has it's count > 0 it means we did not traversed whole substring because we came across a word which was not in temp_hash_map.

Below is the implementation of above approach :

```
 // CPP program to calculate the starting indices
// of substrings inside S which contains all the
// words present in List L.
#include <bits/stdc++.h>
using namespace std;

// Returns an integer vector consisting of starting
// indices of substrings present inside the string S
vector<int> findSubstringIndices(string S,
                            const vector<string>& L)
```

```
{

    // Number of a characters of a word in list L.
    int size_word = L[0].size();

    // Number of words present inside list L.
    int word_count = L.size();

    // Total characters present in list L.
    int size_L = size_word * word_count;

    // Resultant vector which stores indices.
    vector<int> res;

    // If the total number of characters in list L
    // is more than length of string S itself.
    if (size_L > S.size())
        return res;

    // Map stores the words present in list L
    // against it's occurrences inside list L
    unordered_map<string, int> hash_map;

    for (int i = 0; i < word_count; i++)
        hash_map[L[i]]++;

    for (int i = 0; i <= S.size() - size_L; i++) {
        unordered_map<string, int> temp_hash_map(hash_map);

        int j = i;

        // Traverse the substring
        while (j < i + size_L) {

            // Extract the word
            string word = S.substr(j, size_word);

            // If word not found simply break.
            if (hash_map.find(word) == hash_map.end())
                break;

            // Else decrement the count of word from hash_map
            else
                temp_hash_map[word]--;

            j += size_word;
        }
```

```
        int count = 0;
        for (auto itr = temp_hash_map.begin();
             itr != temp_hash_map.end(); itr++)
            if (itr->second > 0)
                count++;

        // Store the starting index of that substring
        if (count == 0)
            res.push_back(i);
    }

    return res;
}

// Driver Code
int main()
{
    string S = "barfoothefoobarman";
    vector<string> L = { "foo", "bar" };
    vector<int> indices = findSubstringIndices(S, L);
    for (int i = 0; i < indices.size(); i++)
        cout << indices[i] << " ";
    return 0;
}
```

Output :

```
0 9
```

Time Complexity : **O(N − K) * K**
N : length of string S.
K : total length of list L if all the words are concatenated. If L : [“ab”, “cd”] then K = 4.

Source

https://www.geeksforgeeks.org/find-starting-indices-substrings-string-s-made-concatenating-words-listl/

Chapter 126

Find three element from different three arrays such that that a + b + c = sum

Find three element from different three arrays such that that a + b + c = sum - GeeksforGeeks

Given three integer arrays and a "sum", the task is to check if there are three elements a, b, c such that a + b + c = sum and a, b and c belong to three different arrays.

Examples :

```
Input : a1[] = { 1 , 2 , 3 , 4 , 5 };
    a2[] = { 2 , 3 , 6 , 1 , 2 };
    a3[] = { 3 , 2 , 4 , 5 , 6 };
        sum = 9
Output : Yes
1  + 2  + 6 = 9  here 1 from a1[] and 2 from
a2[] and 6 from a3[]

Input : a1[] = { 1 , 2 , 3 , 4 , 5 };
    a2[] = { 2 , 3 , 6 , 1 , 2 };
    a3[] = { 3 , 2 , 4 , 5 , 6 };
         sum = 20
Output : No
```

A **naive approach** is to run three loops and check sum of three element form different arrays equal to given number if find then print exist and otherwise print not exist.

C++

```
 // C++ program to find three element
// from different three arrays such
// that that a + b + c is equal to
// given sum
#include<bits/stdc++.h>
using namespace std;

// Function to check if there is
// an element from each array such
// that sum of the three elements
// is equal to given sum.
bool findTriplet(int a1[], int a2[],
                 int a3[], int n1,
                 int n2, int n3, int sum)
{
    for (int i = 0; i < n1; i++)
    for (int j = 0; j < n2; j++)
        for (int k = 0; k < n3; k++)
            if (a1[i] + a2[j] + a3[k] == sum)
            return true;

    return false;
}

// Driver Code
int main()
{
    int a1[] = { 1 , 2 , 3 , 4 , 5 };
    int a2[] = { 2 , 3 , 6 , 1 , 2 };
    int a3[] = { 3 , 2 , 4 , 5 , 6 };
    int sum = 9;
    int n1 = sizeof(a1) / sizeof(a1[0]);
    int n2 = sizeof(a2) / sizeof(a2[0]);
    int n3 = sizeof(a3) / sizeof(a3[0]);
    findTriplet(a1, a2, a3, n1, n2, n3, sum)?
                cout << "Yes" : cout << "No";
    return 0;
}
```

Java

```
 // Java program to find three element
// from different three arrays such
// that that a + b + c is equal to
// given sum
class GFG
{
```

```
    // Function to check if there is
    // an element from each array such
    // that sum of the three elements
    // is equal to given sum.
    static boolean findTriplet(int a1[], int a2[],
                               int a3[], int n1,
                               int n2, int n3, int sum)
    {
        for (int i = 0; i < n1; i++)
            for (int j = 0; j < n2; j++)
                for (int k = 0; k < n3; k++)
                    if (a1[i] + a2[j] + a3[k] == sum)
                    return true;

        return false;
    }

    // Driver code
    public static void main (String[] args)
    {
        int a1[] = { 1 , 2 , 3 , 4 , 5 };
        int a2[] = { 2 , 3 , 6 , 1 , 2 };
        int a3[] = { 3 , 2 , 4 , 5 , 6 };
        int sum = 9;

        int n1 = a1.length;
        int n2 = a2.length;
        int n3 = a3.length;

        if(findTriplet(a1, a2, a3, n1, n2, n3, sum))
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find
# three element from different
# three arrays such that that
# a + b + c is equal to
# given sum

# Function to check if there
# is an element from each
```

```
# array such that sum of the
# three elements is equal to
# given sum.
def findTriplet(a1, a2, a3,
                n1, n2, n3, sum):

    for i in range(0 , n1):
        for j in range(0 , n2):
            for k in range(0 , n3):
                if (a1[i] + a2[j] +
                    a3[k] == sum):
                    return True

    return False

# Driver Code
a1 = [ 1 , 2 , 3 , 4 , 5 ]
a2 = [ 2 , 3 , 6 , 1 , 2 ]
a3 = [ 3 , 2 , 4 , 5 , 6 ]
sum = 9
n1 = len(a1)
n2 = len(a2)
n3 = len(a3)
print("Yes") if findTriplet(a1, a2, a3,
                            n1, n2, n3,
                            sum) else print("No")

# This code is contributed
# by Smitha
```

C#

```
 // C# program to find three element
// from different three arrays such
// that that a + b + c is equal to
// given sum
using System;

public class GFG
{

// Function to check if there is an
// element from each array such that
// sum of the three elements is
// equal to given sum.
static bool findTriplet(int []a1, int []a2,
                        int []a3, int n1,
                        int n2, int n3,
```

```
                        int sum)
{

    for (int i = 0; i < n1; i++)

        for (int j = 0; j < n2; j++)

            for (int k = 0; k < n3; k++)
            if (a1[i] + a2[j] + a3[k] == sum)
            return true;

    return false;
}

    // Driver Code
    static public void Main ()
    {
        int []a1 = {1 , 2 , 3 , 4 , 5};
        int []a2 = {2 , 3 , 6 , 1 , 2};
        int []a3 = {3 , 2 , 4 , 5 , 6};
        int sum = 9;
        int n1 = a1.Length;
        int n2 = a2.Length;
        int n3 = a3.Length;
        if(findTriplet(a1, a2, a3, n1,
                       n2, n3, sum))
        Console.WriteLine("Yes");
        else
        Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find three element
// from different three arrays such
// that that a + b + c is equal to
// given sum

// Function to check if there is an
// element from each array such that
// sum of the three elements is equal
// to given sum.
function findTriplet($a1, $a2, $a3,
                     $n1, $n2, $n3,
```

```
                    $sum)
{
    for ( $i = 0; $i < $n1; $i++)
    for ( $j = 0; $j < $n2; $j++)
        for ( $k = 0; $k < $n3; $k++)
            if ($a1[$i] + $a2[$j] + $a3[$k] == $sum)
            return true;

    return false;
}

// Driver Code
$a1 = array( 1 , 2 , 3 , 4 , 5 );
$a2 = array( 2 , 3 , 6 , 1 , 2 );
$a3 = array( 3 , 2 , 4 , 5 , 6 );
$sum = 9;
$n1 = count($a1);
$n2 = count($a2);
$n3 = count($a3);
if(findTriplet($a1, $a2, $a3, $n1,
               $n2, $n3, $sum)==true)
echo "Yes" ;
else
echo "No";

// This code is contributed by anuj_67.
?>
```

Output :

```
Yes
```

Time complexity : $O(n^3)$
Space complexity : $O(1)$

An **efficient solution** is to store all elements of first array in hash table (unordered_set in C++) and calculate sum of two elements last two array elements one by one and substract from given number k and check in hash table if it's exist in hash table then print exist and otherwise not exist.

```
1. Store all elements of first array in hash table
2. Generate all pairs of elements from two arrays using
   nested loop. For every pair (a1[i], a2[j]), check if
   sum - (a1[i] + a2[j]) exists in hash table. If yes
   return true.
```

Below is C++ implementation of above idea.

C++

```
 // C++ program to find three element
// from different three arrays such
// that that a + b + c is equal to
// given sum
#include<bits/stdc++.h>
using namespace std;

// Function to check if there is
// an element from each array such
// that sum of the three elements is
// equal to given sum.
bool findTriplet(int a1[], int a2[],
                 int a3[], int n1,
                 int n2, int n3,
                 int sum)
{
    // Store elements of
    // first array in hash
    unordered_set <int> s;
    for (int i = 0; i < n1; i++)
        s.insert(a1[i]);

    // sum last two arrays
    // element one by one
    for (int i = 0; i < n2; i++)
    {
        for (int j = 0; j < n3; j++)
        {
            // Consider current pair and
            // find if there is an element
            // in a1[] such that these three
            // form a required triplet
            if (s.find(sum - a2[i] - a3[j]) !=
                                    s.end())
                return true;
        }
    }

    return false;
}

// Driver Code
int main()
{
```

```
    int a1[] = { 1 , 2 , 3 , 4 , 5 };
    int a2[] = { 2 , 3 , 6 , 1 , 2 };
    int a3[] = { 3 , 2 , 4 , 5 , 6 };
    int sum = 9;
    int n1 = sizeof(a1) / sizeof(a1[0]);
    int n2 = sizeof(a2) / sizeof(a2[0]);
    int n3 = sizeof(a3) / sizeof(a3[0]);
    findTriplet(a1, a2, a3, n1, n2, n3, sum)?
    cout << "Yes" : cout << "No";

    return 0;
}
```

Output :

```
Yes
```

Time complexity : O(n^2)
Auxiliary Space : O(n)

References :
http://stackoverflow.com/questions/2070359/finding-three-elements-in-an-array-whose-sum-is-closest-to-a-given-n

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-three-element-from-different-three-arrays-such-that-that-a-b-c-k/

Chapter 127

Find top k (or most frequent) numbers in a stream

Find top k (or most frequent) numbers in a stream - GeeksforGeeks

Given an array of n numbers. Your task is to read numbers from the array and keep at-most K numbers at the top (According to their decreasing frequency) every time a new number is read. We basically need to print top k numbers sorted by frequency when input stream has included k distinct elements, else need to print all distinct elements sorted by frequency.

Examples:

```
Input :  arr[] = {5, 2, 1, 3, 2}
             k = 4
Output : 5 2 5 1 2 5 1 2 3 5 2 1 3 5

Input  : arr[] = {5, 2, 1, 3, 4}
             k = 4
Output : 5 2 5 1 2 5 1 2 3 5 1 2 3 4
```

Expected time complexity is O(n * k)

Explanation of 1st example:
Given array is arr[] = {5, 2, 1, 3, 2} and k = 4
Step 1:After reading 5, there is only one element 5 whose frequency is max till now. so print 5.
Step 2:After reading 2, we will have two elements 2 and 5 with same frequency. As 2, is smaller than 5 but their frequency is same so we will print 2 5.
Step 3: After reading 1, we will have 3 elements 1,2 and 5 with same frequency, so print 1 2 5.
Step 4: Similarly after reading 3, print 1 2 3 5
Step 5: After reading last element 2, since 2 has already occurred so we have now frequency

of 2 as 2. So we keep 2 at the top and then rest of element with same frequency in sorted order. So print, 2 1 3 5.

Below is the step by step algorithm to do this:

1. Iterate through the array which contains stream of numbers.
2. To keep track of top k elements, make a top vector of size k+1.
3. For every element in the stream increase its frequency and store it in the last position of top vector. We can use hashing for efficiently fetching frequency of an element and increasing it.
4. Now find the position of element in top vector and iterate from that position to zero. For finding position we can make use of the find() function in C++ STL, it returns an iterator pointing to element if found in the vector.
5. And make that list of k+1 numbers sorted according to frequency and their value.
6. Print top k elements form top vector.
7. Repeat the above steps for every element in the stream.

Below is the C++ implementation of above idea:

C++

```
 // C++ program to find top k elements in a stream
#include <bits/stdc++.h>
using namespace std;

// Function to print top k numbers
void kTop(int a[], int n, int k)
{
    // vector of size k+1 to store elements
    vector<int> top(k + 1);

    // array to keep track of frequency
    unordered_map<int, int> freq;

    // iterate till the end of stream
    for (int m = 0; m < n; m++)
    {
        // increase the frequency
        freq[a[m]]++;

        // store that element in top vector
        top[k] = a[m];

        // search in top vector for same element
        auto it = find(top.begin(), top.end() - 1, a[m]);

        // iterate from the position of element to zero
```

```
        for (int i = distance(top.begin(), it) - 1; i >= 0; --i)
        {
            // compare the frequency and swap if higher
            // frequency element is stored next to it
            if (freq[top[i]] < freq[top[i + 1]])
                swap(top[i], top[i + 1]);

            // if frequency is same compare the elements
            // and swap if next element is high
            else if ((freq[top[i]] == freq[top[i + 1]])
                     && (top[i] > top[i + 1]))
                swap(top[i], top[i + 1]);
            else
                break;
        }

        // print top k elements
        for (int i = 0; i < k && top[i] != 0; ++i)
            cout << top[i] << ' ';
    }
    cout << endl;
}

// Driver program to test above function
int main()
{
    int k = 4;
    int arr[] = { 5, 2, 1, 3, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);
    kTop(arr, n, k);
    return 0;
}
```

Python

```
 # Python program to find top k elements in a stream

# Function to print top k numbers
def kTop(a, n, k):

    # list of size k+1 to store elements
    top = [0 for i in range(k + 1)]

    # dictionary to keep track of frequency
    freq = {i:0 for i in range(k + 1)}

    # iterate till the end of stream
    for m in range(n):
```

```
        # increase the frequency
        if a[m] in freq.keys():
            freq[a[m]] += 1
        else:
            freq[a[m]] = 1

        # store that element in top vector
        top[k] = a[m]

        i = top.index(a[m])
        i -= 1

        while i >= 0:

            # compare the frequency and swap if higher
            # frequency element is stored next to it
            if (freq[top[i]] < freq[top[i + 1]]):
                t = top[i]
                top[i] = top[i + 1]
                top[i + 1] = t

            # if frequency is same compare the elements
            # and swap if next element is high
            elif ((freq[top[i]] == freq[top[i + 1]]) and (top[i] > top[i + 1])):
                t = top[i]
                top[i] = top[i + 1]
                top[i + 1] = t
            else:
                break
            i -= 1

        # print top k elements
        i = 0
        while i < k and top[i] != 0:
            print top[i],
            i += 1
    print

# Driver program to test above function
k = 4
arr = [ 5, 2, 1, 3, 2 ]
n = len(arr)
kTop(arr, n, k)

# This code is contributed by Sachin Bisht
```

Output:

```
5 2 5 1 2 5 1 2 3 5 2 1 3 5
```

Time Complexity: O(n * k)

Source

https://www.geeksforgeeks.org/find-top-k-or-most-frequent-numbers-in-a-stream/

Chapter 128

Find top three repeated in array

Find top three repeated in array - GeeksforGeeks

Given an array of size N with repeated numbers, You Have to Find the top three repeated numbers.
Note : If Number comes same number of times then our output is one who comes first in array
Examples:

> Input : arr[] = {3, 4, 2, 3, 16, 3, 15, 16, 15, 15, 16, 2, 3}
> Output : Three largest elements are 3 16 15
> Explanation :Here, 3 comes 4 times, 16 comes 3 times, 15 comes 3 times.
>
> Input : arr[] = {2, 4, 3, 2, 3, 4, 5, 5, 3, 2, 2, 5}
> Output : Three largest elements are 2 3 5

Asked in :Zoho

First We have to find the frequency of each element in a hash table **freq**. Now Our Task is to Find top 3 elements in the hash table, To Find it We just use three pair type variable (suppose x, y, z)in which first store the frequency and second store the actual number.
Algorithm

```
1) Initialize the largest three elements
   as Minimum value.
    x.first = y.first = z.first = Minus-Infinite

2) Iterate through all elements of the
   hash table freq.
   a) Let current array element be p.
   b) If (fre[p] !=0 && fre[p] > x.first)
      {
          // This order of assignment is important
```

```
          z = y
          y = x
          x.first = fre[p]
          x.second = p;
       }
   c) Else if (fre[p] !=0 && free[p] > y.first)
      {
          z = y
          y.first = fre[p]
          y.second = p
      }
   d) Else if (fre[p] !=0 && free[p] > z.first)
      {
          z.first = fre[p]
          z.second = p
      }

// Modify frequency of Current element
// as zero because We Traverse Initial
// array arr[]. So it don't take same
// values again
3) fre[p] = 0

3) Print x.second, y.second and z.second.


 // C++ Program to Find the top three repeated numbers
#include <bits/stdc++.h>
using namespace std;

/* Function to print top three repeated numbers */
void top3Repeated(int arr[], int n)
{
    // There should be atleast two elements
    if (n < 3) {
        cout << "Invalid Input";
        return;
    }

    // Count Frequency of each element
    unordered_map<int, int> fre;
    for (int i = 0; i < n; i++)
        fre[arr[i]]++;

    // Initialize first value of each variable
    // of Pair type is INT_MIN
    pair<int, int> x, y, z;
    x.first = y.first = z.first = INT_MIN;
```

```
for (auto curr : fre) {

    // If frequency of current element
    // is not zero and greater than
    // frequency of first largest element
    if (curr.second > x.first) {

        // Update second and third largest
        z = y;
        y = x;

        // Modify values of x Number
        x.first = curr.second;
        x.second = curr.first;
    }

    // If frequency of current element is
    // not zero and frequency of current
    // element is less than frequency of
    // first largest element, but greater
    // than y element
    else if (curr.second > y.first) {

        // Modify values of third largest
        z = y;

        // Modify values of second largest
        y.first = curr.second;
        y.second = curr.first;
    }

    // If frequency of current element
    // is not zero and frequency of
    // current element is less than
    // frequency of first element and
    // second largest, but greater than
    // third largest.
    else if (curr.second > z.first) {

        // Modify values of z Number
        z.first = curr.second;
        z.second = curr.first;
    }
}

cout << "Three largest elements are "
    << x.second << " " << y.second
    << " " << z.second;
```

```
}

// Driver's Code
int main()
{
    int arr[] = { 3, 4, 2, 3, 16, 3, 15,
                  16, 15, 15, 16, 2, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    top3Repeated(arr, n);
    return 0;
}
```

Output:

```
Three largest elements are 3 15 16
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/find-top-three-repeated-array/

Chapter 129

Find uncommon characters of the two strings

Find uncommon characters of the two strings - GeeksforGeeks

Find and print the uncommon characters of the two given strings in sorted order. Here uncommon character means that either the character is present in one string or it is present in other string but not in both. The strings contain only lowercase characters and can contain duplicates.

Source: Amazon Interview Experience | Set 355 (For 1 Year Experienced)

Examples:

```
Input : str1 = "characters"
        str2 = "alphabets"
Output : b c l p r

Input : str1 = "geeksforgeeks"
        str2 = "geeksquiz"
Output : f i o q r u z
```

Naive Approach: Using two loops. For each charactrer of 1st string check whether it is present in 2nd string or not. Likewise, for each character of 2nd string check whether it is present in 1st string or not. Time Complexity $O(n_2)$ and extra would be required to handle duplicates.

Efficient Approach: Use hashing. Use a hash table of size 26 for all the lowercase characters.
Initially, mark presence of each character as '0' (denoting that the character is not present in both the strings). Traverse the 1st string and mark presence of each character of 1st string as '1' (denoting 1st string) in the hash table. Now, traverse the 2nd string. For each character of 2nd string, check whether its presence in the hash table is '1' or not. If it is '1',

then mark its presence as '-1' (denoting that the character is common to both the strings), else mark its presence as '2' (denoting 2nd string).

```
 // C++ implementation to find the uncommon
// characters of the two strings
#include <bits/stdc++.h>
using namespace std;

// size of the hash table
const int MAX_CHAR = 26;

// function to find the uncommon characters
// of the two strings
void findAndPrintUncommonChars(string str1, string str2)
{
    // mark presence of each character as 0
    // in the hash table 'present[]'
    int present[MAX_CHAR];
    for (int i=0; i<MAX_CHAR; i++)
        present[i] = 0;

    int l1 = str1.size();
    int l2 = str2.size();

    // for each character of str1, mark its
    // presence as 1 in 'present[]'
    for (int i=0; i<l1; i++)
        present[str1[i] - 'a'] = 1;

    // for each character of str2
    for (int i=0; i<l2; i++)
    {
        // if a character of str2 is also present
        // in str1, then mark its presence as -1
        if (present[str2[i] - 'a'] == 1
            || present[str2[i] - 'a'] == -1)
            present[str2[i] - 'a'] = -1;

        // else mark its presence as 2
        else
            present[str2[i] - 'a'] = 2;
    }

    // print all the uncommon characters
    for (int i=0; i<MAX_CHAR; i++)
        if (present[i] == 1 || present[i] == 2 )
            cout << (char(i + 'a')) << " ";
}
```

```
// Driver program to test above
int main()
{
    string str1 = "characters";
    string str2 = "alphabets";
    findAndPrintUncommonChars(str1, str2);
    return 0;
}
```

Output:

```
b c l p r
```

Time Complexity: O(m + n), where **m** and **n** are the sizes of the two strings respectively.

Source

https://www.geeksforgeeks.org/find-uncommon-characters-two-strings/

Chapter 130

Find unique elements in linked list

Find unique elements in linked list - GeeksforGeeks

Given a linked list. We need to find unique elements in the linked list i.e, those elements which are not repeated in the linked list or those elements whose frequency is 1. If No such elements are present in list so Print " No Unique Elements".

Examples:

```
Input : 1 -> 4 -> 4 -> 2 -> 3 -> 5 -> 3 -> 4 -> 5
Output :1 2

Input :4 -> 5 -> 2 -> 5 -> 1 -> 4 -> 1 -> 2
Output :No Unique Elements
```

Method 1 (Using Two Loops) This is the simple way where two loops are used. Outer loop is used to pick the elements one by one and inner loop compares the picked element with rest of the elements. If Element is not equal to other elements than Print that Element. Time Complexity : O(N * n)

Method 2 (Sorting) : Sort the elements using Merge Sort. O(n Log n). Now Traverse List in linear time and check if current element is not equal to previous element then Print O(N)
Please note that this method doesn't preserve the original order of elements.

Time Complexity: O(NLogN)

Method 3 (Hashing)
We use the concept of Hash table Here, We traverse the link list from head to end. For every newly encountered element, we put it in the hash table after that we again traverse list and Print those elements whose frequency is 1.Time Complexity : O(N)
Below is the Implementation of this

```
 // C++ Program to Find the Unique elements in
// linked lists
#include <bits/stdc++.h>
using namespace std;

/* Linked list node */
struct Node {
    int data;
    struct Node* next;
};

/* Function to insert a node at the beginning of
   the linked list */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = *head_ref;
    *head_ref = new_node;
}

// function to Find the unique elements in linked lists
void uniqueElements(struct Node* head)
{
    // Initialize hash array that store the
    // frequency of each element of list
    unordered_map<int, int> hash;

    for (Node *temp=head; temp!=NULL; temp=temp->next)
        hash[temp->data]++;

    int count = 0;
    for (Node *temp=head; temp!=NULL; temp=temp->next) {

        // Check whether the frequency of current
        // element is 1 or not
        if (hash[temp->data] == 1) {
            cout << temp->data << " ";
            count++;
        }
    }

    // If No unique element in list
    if (count == 0)
        cout << " No Unique Elements ";
}

// Driver program to test above
```

```
int main()
{
    struct Node* head = NULL;

    // creating linked list
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 5);
    push(&head, 3);
    push(&head, 2);
    push(&head, 4);
    push(&head, 4);
    push(&head, 1);
    uniqueElements(head);
    return 0;
}
```

Output:

```
1 2
```

Time Complexity : O(N)
Auxiliary Space : O(N)

Source

https://www.geeksforgeeks.org/find-unique-elements-linked-list/

Chapter 131

Find whether an array is subset of another array | Added Method 3

Find whether an array is subset of another array | Added Method 3 - GeeksforGeeks

Given two arrays: arr1[0..m-1] and arr2[0..n-1]. Find whether arr2[] is a subset of arr1[] or not. Both the arrays are not in sorted order. It may be assumed that elements in both array are distinct.

Examples:
Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}
Output: arr2[] is a subset of arr1[]

Input: arr1[] = {1, 2, 3, 4, 5, 6}, arr2[] = {1, 2, 4}
Output: arr2[] is a subset of arr1[]

Input: arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}
Output: arr2[] is not a subset of arr1[]

Method 1 (Simple)
Use two loops: The outer loop picks all the elements of arr2[] one by one. The inner loop linearly searches for the element picked by outer loop. If all elements are found then return 1, else return 0.

C++

```
 #include<bits/stdc++.h>

/* Return 1 if arr2[] is a subset of
arr1[] */
bool isSubset(int arr1[], int arr2[],
                        int m, int n)
{
```

```
    int i = 0;
    int j = 0;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            if(arr2[i] == arr1[j])
                break;
        }

        /* If the above inner loop was
        not broken at all then arr2[i]
        is not present in arr1[] */
        if (j == m)
            return 0;
    }

    /* If we reach here then all
    elements of arr2[] are present
    in arr1[] */
    return 1;
}

// Driver code
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
        printf("arr2[] is subset of arr1[] ");
    else
        printf("arr2[] is not a subset of arr1[]");

    getchar();
    return 0;
}
```

Java

```
 // Java program to find whether an array
// is subset of another array

class GFG {
```

```
    /* Return true if arr2[] is a subset
    of arr1[] */
    static boolean isSubset(int arr1[],
                int arr2[], int m, int n)
    {
        int i = 0;
        int j = 0;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                if(arr2[i] == arr1[j])
                    break;

            /* If the above inner loop
            was not broken at all then
            arr2[i] is not present in
            arr1[] */
            if (j == m)
                return false;
        }

        /* If we reach here then all
        elements of arr2[] are present
        in arr1[] */
        return true;
    }

    // Driver code
    public static void main(String args[])
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
            System.out.print("arr2[] is "
                  + "subset of arr1[] ");
        else
            System.out.print("arr2[] is "
             + "not a subset of arr1[]");
    }
}
```

C#

```
 // C# program to find whether an array
```

```
// is subset of another array
using System;

class GFG {

    /* Return true if arr2[] is a
    subset of arr1[] */
    static bool isSubset(int []arr1,
               int []arr2, int m, int n)
    {
        int i = 0;
        int j = 0;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                if(arr2[i] == arr1[j])
                    break;

            /* If the above inner loop
            was not broken at all then
            arr2[i] is not present in
            arr1[] */
            if (j == m)
                return false;
        }

        /* If we reach here then all
        elements of arr2[] are present
        in arr1[] */
        return true;
    }

    // Driver function
    public static void Main()
    {
        int []arr1 = {11, 1, 13, 21, 3, 7};
        int []arr2 = {11, 3, 7, 1};

        int m = arr1.Length;
        int n = arr2.Length;

        if(isSubset(arr1, arr2, m, n))
        Console.WriteLine("arr2[] is subset"
                          + " of arr1[] ");
        else
        Console.WriteLine("arr2[] is not a "
                      + "subset of arr1[]");
    }
```

```
}

// This code is contributed by Sam007
```

PHP

```
 <?php
/* Return 1 if arr2[] is a subset of
arr1[] */
function isSubset($arr1, $arr2, $m, $n)
{
    $i = 0;
    $j = 0;
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $m; $j++)
        {
            if($arr2[$i] == $arr1[$j])
                break;
        }

        /* If the above inner loop was
        not broken at all then arr2[i]
        is not present in arr1[] */
        if ($j == $m)
            return 0;
    }

    /* If we reach here then all
    elements of arr2[] are present
    in arr1[] */
    return 1;
}

// Driver code
    $arr1 = array(11, 1, 13, 21, 3, 7);
    $arr2 = array(11, 3, 7, 1);

    $m = count($arr1);
    $n = count($arr2);

    if(isSubset($arr1, $arr2, $m, $n))
        echo "arr2[] is subset of arr1[] ";
    else
        echo "arr2[] is not a subset of arr1[]";

// This code is contributed by anuj_67.
?>
```

Output:

```
arr2[] is subset of arr1[]
```

Time Complexity: O(m*n)

Method 2 (Use Sorting and Binary Search)

```
1) Sort arr1[] O(mLogm)
2) For each element of arr2[], do binary search for it in sorted arr1[].
         a) If the element is not found then return 0.
3) If all elements are present then return 1.
```

C

```
 #include<stdio.h>

/* Fucntion prototypes */
void quickSort(int *arr, int si, int ei);
int binarySearch(int arr[], int low, int high, int x);

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0;

    quickSort(arr1, 0, m-1);
    for (i=0; i<n; i++)
    {
        if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
           return 0;
    }

    /* If we reach here then all elements of arr2[]
      are present in arr1[] */
    return 1;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int x)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
```

```
    /* Check if arr[mid] is the first occurrence of x.
        arr[mid] is first occurrence if x is one of the following
        is true:
        (i)  mid == 0 and arr[mid] == x
        (ii) arr[mid-1] < x and arr[mid] == x
     */
    if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))
      return mid;
    else if(x > arr[mid])
      return binarySearch(arr, (mid + 1), high, x);
    else
      return binarySearch(arr, low, (mid -1), x);
  }
 return -1;
}

void exchange(int *a, int *b)
{
    int temp;
    temp = *a;
    *a   = *b;
    *b   = temp;
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++)
    {
        if(A[j] <= x)
        {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange (&A[i + 1], &A[ei]);
    return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
```

```
void quickSort(int A[], int si, int ei)
{
    int pi;    /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
      printf("arr2[] is subset of arr1[] ");
    else
      printf("arr2[] is not a subset of arr1[] ");

    return 0;
}
```

Java

```
 class Main
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m, int n)
    {
        int i = 0;

        sort(arr1, 0, m-1);
        for (i=0; i<n; i++)
        {
            if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
               return false;
        }

        /* If we reach here then all elements of arr2[]
          are present in arr1[] */
        return true;
    }
```

```
/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
static int binarySearch(int arr[], int low, int high, int x)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/

    /* Check if arr[mid] is the first occurrence of x.
        arr[mid] is first occurrence if x is one of the following
        is true:
        (i)  mid == 0 and arr[mid] == x
        (ii) arr[mid-1] < x and arr[mid] == x
     */
    if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))
      return mid;
    else if(x > arr[mid])
      return binarySearch(arr, (mid + 1), high, x);
    else
      return binarySearch(arr, low, (mid -1), x);
  }
 return -1;
}

/* This function takes last element as pivot,
   places the pivot element at its correct
   position in sorted array, and places all
   smaller (smaller than pivot) to left of
   pivot and all greater elements to right
   of pivot */
static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<high; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
```

```
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
      high  --> Ending index */
    static void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    public static void main(String args[])
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
          System.out.print("arr2[] is subset of arr1[] ");
        else
          System.out.print("arr2[] is not a subset of arr1[]");
    }
}
```

Time Complexity: O(mLogm + nLogm). Please note that this will be the complexity if an mLogm algorithm is used for sorting which is not the case in above code. In above code Quick Sort is sued and worst case time complexity of Quick Sort is O(m^2)

Method 3 (Use Sorting and Merging)
1) Sort both arrays: arr1[] and arr2[] O(mLogm + nLogn)
2) Use Merge type of process to see if all elements of sorted arr2[] are present in sorted arr1[].

Thanks to **Parthsarthi** for suggesting this method.

C++

```
 #include <bits/stdc++.h>
using namespace std;

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0, j = 0;

    if (m < n)
       return 0;

    sort(arr1, arr1 + m);
    sort(arr2, arr2 + n);
    while (i < n && j < m )
    {
        if( arr1[j] <arr2[i] )
            j++;
        else if( arr1[j] == arr2[i] )
        {
            j++;
            i++;
        }
        else if( arr1[j] > arr2[i] )
            return 0;
    }

    return  (i < n)? false : true;
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
```

```
      printf("arr2[] is subset of arr1[] ");
    else
      printf("arr2[] is not a subset of arr1[] ");

    return 0;
}
```

Java

```
 // Java code to find whether an array is subset of
// another array
import java.util.Arrays;
class GFG
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m,
                                                    int n)
    {
        int i = 0, j = 0;

        if(m < n)
        return false;

        Arrays.sort(arr1); //sorts arr1
        Arrays.sort(arr2); // sorts arr2

        while( i < n && j < m )
        {
            if( arr1[j] < arr2[i] )
                j++;
            else if( arr1[j] == arr2[i] )
            {
                j++;
                i++;
            }
            else if( arr1[j] > arr2[i] )
                return false;
        }

        if( i < n )
            return false;
        else
            return true;
    }

    public static void main(String[] args)
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
```

```
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
        System.out.println("arr2 is a subset of arr1");
        else
        System.out.println("arr2 is not a subset of arr1");
    }
}
// This code is contributed by Kamal Rawal
```

C#

```
 // C# code to find whether an array
// is subset of another array
using System;
class GFG {

    // Return true if arr2[] is
    // a subset of arr1[] */
    static bool isSubset(int []arr1,
                         int []arr2,
                         int m,
                         int n)
    {
        int i = 0, j = 0;

        if(m < n)
            return false;

        //sorts arr1
        Array.Sort(arr1);

        // sorts arr2
        Array.Sort(arr2);

        while( i < n && j < m )
        {
            if( arr1[j] < arr2[i] )
                j++;
            else if( arr1[j] == arr2[i] )
            {
                j++;
                i++;
            }
            else if( arr1[j] > arr2[i] )
```

```
                return false;
        }

        if( i < n )
            return false;
        else
            return true;
    }

    // Driver Code
    public static void Main()
    {
        int []arr1 = {11, 1, 13, 21, 3, 7};
        int []arr2 = {11, 3, 7, 1};

        int m = arr1.Length;
        int n = arr2.Length;

        if(isSubset(arr1, arr2, m, n))
            Console.Write("arr2 is a subset of arr1");
        else
            Console.Write("arr2 is not a subset of arr1");
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php

/* Return 1 if arr2[] is a subset of arr1[] */
function isSubset( $arr1, $arr2, $m, $n)
{
    $i = 0; $j = 0;

    if ($m < $n)
        return 0;

    sort($arr1);
    sort($arr2);

    while ($i < $n and $j < $m )
    {
        if( $arr1[$j] <$arr2[$i] )
            $j++;
        else if( $arr1[$j] == $arr2[$i] )
        {
```

```
            $j++;
            $i++;
        }
        else if( $arr1[$j] > $arr2[$i] )
            return 0;
    }

    return ($i < $n) ? false : true;
}

/*Driver program to test above functions */

    $arr1 = array(11, 1, 13, 21, 3, 7);
    $arr2 = array(11, 3, 7, 1);

    $m = count($arr1);
    $n = count($arr2);

    if(isSubset($arr1, $arr2, $m, $n))
        echo "arr2[] is subset of arr1[] ";
    else
        echo "arr2[] is not a subset of arr1[] ";

// This code is contributed by anuj_67.
?>
```

Output:

```
arr2 is a subset of arr1
```

Time Complexity: O(mLogm + nLogn) which is better than method 2. Please note that this will be the complexity if an nLogn algorithm is used for sorting both arrays which is not the case in above code. In above code Quick Sort is sued and worst case time complexity of Quick Sort is O(n^2)

Method 4 (Use Hashing)
1) Create a Hash Table for all the elements of arr1[].
2) Traverse arr2[] and search for each element of arr2[] in the Hash Table. If element is not found then return 0.
3) If all elements are found then return 1.

```
 // Java code to find whether an array is subset of
// another array
import java.util.HashSet;
class GFG
{
    /* Return true if arr2[] is a subset of arr1[] */
```

```
    static boolean isSubset(int arr1[], int arr2[], int m,
                                                    int n)
    {
        HashSet<Integer> hset= new HashSet<>();

        // hset stores all the values of arr1
        for(int i = 0; i < m; i++)
        {
            if(!hset.contains(arr1[i]))
                hset.add(arr1[i]);
        }

        // loop to check if all elements of arr2 also
        // lies in arr1
        for(int i = 0; i < n; i++)
        {
            if(!hset.contains(arr2[i]))
                return false;
        }
        return true;
    }

    public static void main(String[] args)
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
        System.out.println("arr2 is a subset of arr1");
        else
        System.out.println("arr2 is not a subset of arr1");
    }
}
// This code is contributed by Kamal Rawal
```

Note that method 1, method 2 and method 4 don't handle the cases when we have duplicates in arr2[]. For example, {1, 4, 4, 2} is not a subset of {1, 4, 2}, but these methods will print it as a subset.

Improved By : Sam007, nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/

Chapter 132

Find winner of an election where votes are represented as candidate names

Find winner of an election where votes are represented as candidate names - GeeksforGeeks

Given an array of names of candidates in an election. A candidate name in array represents a vote casted to the candidate. Print the name of candidates received Max vote. If there is tie, print lexicographically smaller name.

Examples:

```
Input :  votes[] = {"john", "johnny", "jackie",
                    "johnny", "john", "jackie",
                    "jamie", "jamie", "john",
                    "johnny", "jamie", "johnny",
                    "john"};
Output : John
We have four Candidates with name as 'John',
'Johnny', 'jamie', 'jackie'. The candidates
John and Johny get maximum votes. Since John
is alphabetically smaller, we print it.
```

A **simple solution** is to run two loops and count occurrences of every word. Time complexity of this solution is O(n * n * MAX_WORD_LEN).

An **efficient solution** is to use Hashing. We insert all votes in a hash map and keep track of counts of different names. Finally we traverse the map and print the person with maximum votes.

```
// Java program to find winner in an election.
```

```
import java.util.*;

public class ElectoralVotingBallot
{
    /* We have four Candidates with name as 'John',
      'Johnny', 'jamie', 'jackie'.
       The votes in String array are as per the
       votes casted. Print the name of candidates
       received Max vote. */
    public static void findWinner(String votes[])
    {
        // Insert all votes in a hashmap
        Map<String,Integer> map =
                    new HashMap<String, Integer>();
        for (String str : votes)
        {
            if (map.keySet().contains(str))
                map.put(str, map.get(str) + 1);
            else
                map.put(str, 1);
        }

        // Traverse through map to find the candidate
        // with maximum votes.
        int maxValueInMap = 0;
        String winner = "";
        for (Map.Entry<String,Integer> entry : map.entrySet())
        {
            String key  = entry.getKey();
            Integer val = entry.getValue();
            if (val > maxValueInMap)
            {
                maxValueInMap = val;
                winner = key;
            }

            // If there is a tie, pick lexicographically
            // smaller.
            else if (val == maxValueInMap &&
                winner.compareTo(key) > 0)
                winner = key;
        }
        System.out.println(winner);
    }

    // Driver code
    public static void main(String[] args)
    {
```

```
        String[] votes = { "john", "johnny", "jackie",
                           "johnny", "john", "jackie",
                           "jamie", "jamie", "john",
                           "johnny", "jamie", "johnny",
                           "john" };

        findWinner(votes);
    }
}
```

Output:

```
John
```

Another efficient solution is to use Trie. Please refer most frequent word in an array of strings.

Improved By : Ankit Verma 10

Source

https://www.geeksforgeeks.org/find-winner-election-votes-represented-candidate-names/

Chapter 133

First element occurring k times in an array

First element occurring k times in an array - GeeksforGeeks

Given an array of n integers. The task is to find the first element that occurs k number of times. If no element occurs k times the print -1. The distribution of integer elements could be in any range.

Examples:

```
Input : {1, 7, 4, 3, 4, 8, 7},
        k = 2
Output : 7
Both 7 and 4 occur 2 times.
But 7 is the first that occurs 2 times.

Input : {4, 1, 6, 1, 6, 4},
        k = 1
Output : -1
```

Simple Approach: By using two loops. It has a time complexity of $O(n^2)$.

Efficient Approach: Use unordered_map for hashing as range is not known. Steps:

1. Traverse the array elements from left to right.
2. While traversing increment their count in the hash table.
3. Again traverse the array from left to right and check which element has a count equal to k. Print that element and stop.
4. If no element has a count equal to k, print -1.

C++

```
 // C++ implementation to find first
// element occurring k times
#include <bits/stdc++.h>

using namespace std;

// function to find the first element
// occurring k number of times
int firstElement(int arr[], int n, int k)
{
    // unordered_map to count
    // occurrences of each element
    unordered_map<int, int> count_map;
    for (int i=0; i<n; i++)
        count_map[arr[i]]++;

    for (int i=0; i<n; i++)

        // if count of element == k ,then
        // it is the required first element
        if (count_map[arr[i]] == k)
            return arr[i];

    // no element occurs k times
    return -1;
}

// Driver program to test above
int main()
{
    int arr[] = {1, 7, 4, 3, 4, 8, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    cout << firstElement(arr, n, k);
    return 0;
}
```

Python3

```
 # Python implementation to
# find first element
# occurring k times

# function to find the
# first element occurring
# k number of times
def firstElement(arr, n, k):
```

```
    # dictionary to count
    # occurrences of
    # each element
    count_map = {};
    for i in range(0, n):
        if(arr[i] in count_map.keys()):
            count_map[arr[i]] += 1
        else:
            count_map[arr[i]] = 1
        i += 1

    for i in range(0, n):

        # if count of element == k ,
        # then it is the required
        # first element
        if (count_map[arr[i]] == k):
            return arr[i]
        i += 1

    # no element occurs k times
    return -1

# Driver Code
if __name__=="__main__":

    arr = [1, 7, 4, 3, 4, 8, 7];
    n = len(arr)
    k = 2
    print(firstElement(arr, n, k))

# This code is contributed
# by Abhishek Sharma
```

Output:

```
7
```

Time Complexity: O(n)

Improved By : Abhishekk781

Source

https://www.geeksforgeeks.org/first-element-occurring-k-times-array/

Chapter 134

First non-repeating in a linked list

First non-repeating in a linked list - GeeksforGeeks

Given a linked list, find its first non-repeating integer element.

Examples:

```
Input : 10->20->30->10->20->40->30->NULL
Output :First Non-repeating element is 40.

Input :1->1->2->2->3->4->3->4->5->NULL
Output :First Non-repeating element is 5.

Input :1->1->2->2->3->4->3->4->NULL
Output :No NOn-repeating element is found.
```

1) Create a hash table and marked all element as zero.
2) Traverse the linked list and count the frequency of all the element in hashtable.
3) Traverse the linked list again and see the element who's frequency is 1 in hashtable.

```
 // C++ program to find first non-repeating
// element in a linked list
#include<bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
```

```
};

/* Function to find the first non-repeating
 element in  the linked list */
int firstNonRepeating(struct Node *head)
{
    // Create an empty map and insert all linked
    // list elements into hash table
    unordered_map<int, int> mp;
    for (Node *temp=head; temp!=NULL; temp=temp->next)
       mp[temp->data]++;

    // Traverse the linked list again and return
    // the first node whose count is 1
    for (Node *temp=head; temp!=NULL; temp=temp->next)
       if (mp[temp->data] == 1)
           return temp->data;

    return -1;
}

/* Function to push a node */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
            (struct Node*) malloc(sizeof(struct Node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}

/* Driver program to test above function*/
int main()
{
     // Let us create below linked list.
     // 85->15->18->20->85->35->4->20->NULL
     struct Node* head = NULL;
     push(&head, 20);
     push(&head, 4);
     push(&head, 35);
     push(&head, 85);
     push(&head, 20);
     push(&head, 18);
     push(&head, 15);
     push(&head, 85);
     cout << firstNonRepeating(head);
     return 0;
}
```

Output:

```
15
```

Further Optimizations:
The above solution requires two traversals of linked list. In case we have many repeating elements, we can save one traversal by storing positions also in hash table. Please refer last method of Given a string, find its first non-repeating character for details.

Source

https://www.geeksforgeeks.org/first-non-repeating-linked-list/

Chapter 135

For each element in 1st array count elements less than or equal to it in 2nd array | Set 2

For each element in 1st array count elements less than or equal to it in 2nd array | Set 2 - GeeksforGeeks

Given two unsorted arrays arr1[] and arr2[]. They may contain duplicates. For each element in arr1[] count elements less than or equal to it in array arr2[].

Examples:

```
Input : arr1[] = [1, 2, 3, 4, 7, 9]
        arr2[] = [0, 1, 2, 1, 1, 4]
Output : [4, 5, 5, 6, 6, 6]

Input : arr1[] = [5, 10, 2, 6, 1, 8, 6, 12]
        arr2[] = [6, 5, 11, 4, 2, 3, 7]
Output : [4, 6, 1, 5, 0, 6, 5, 7]
```

This problem is already discussed in the previous post. In this article, a more optimized linear time solution to the above problem is discussed. The approach discussed here works for arrays with values in small range. A range of values that can be used as index in an array.

The idea is to use an array to create a direct address tableinitially filled with zero, such that hash[i] gives the count of an element i in the second array arr2[]. Now, calculate pre-sum of the hash array. Doing this, hash[i] will now give the count of elements less than or equal to i in second array arr2[].

Now, traverse the first array and print hash[arr1[i]].

Below program illustrate the above approach:

C++

```
 // C++ program for each element in 1st
// array count elements less than or equal to it
// in 2nd array

#include <iostream>
using namespace std;

#define MAX 100000

// Function for each element in 1st
// array count elements less than or equal to it
// in 2nd array
void countEleLessThanOrEqual(int arr1[], int m,
                             int arr2[], int n)
{
    // Creating hash array initially
    // filled with zero
    int hash[MAX] = {0};

    // Insert element of arr2[] to hash
    // such that hash[i] will give count of
    // element i in arr2[]
    for (int i = 0; i < n; i++)
        hash[arr2[i]]++;

    // Presum of hash array
    // such that hash[i] will give count of
    // element less than or equals to i in arr2[]
    for (int i=1; i<MAX; i++)
        hash[i] = hash[i] + hash[i-1];

    // Traverse arr1[] and print hash[arr[i]]
    for (int i = 0; i < m; i++)
        cout << hash[arr1[i]] << " ";
}

// Driver code
int main()
{
    int arr1[] = {1, 2, 3, 4, 7, 9};
    int arr2[] = {0, 1, 2, 1, 1, 4};
    int m, n;
```

```
    m = sizeof(arr1) / sizeof(arr1[0]);
    n = sizeof(arr2) / sizeof(arr2[0]);

    countEleLessThanOrEqual(arr1, m, arr2, n);

    return 0;
}
```

Java

```
 // Java program for each element
// in 1st array count elements
// less than or equal to it in
// 2nd array
import java.io.*;

class GFG
{
static int MAX = 100000;

// Function for each element
// in 1st array count elements
// less than or equal to it
// in 2nd array
static void countEleLessThanOrEqual(int arr1[], int m,
                                    int arr2[], int n)
{
    // Creating hash array initially
    // filled with zero
    int hash[] = new int[MAX];

    // Insert element of arr2[] to
    // hash such that hash[i] will
    // give count of element i in arr2[]
    for (int i = 0; i < n; i++)
        hash[arr2[i]]++;

    // Presum of hash array
    // such that hash[i] will
    // give count of element
    // less than or equals to
    // i in arr2[]
    for(int i = 1; i < MAX; i++)
    {
        hash[i] = hash[i] +
                  hash[i - 1];
    }
```

```
    // Traverse arr1[] and
    // print hash[arr[i]]
    for (int i = 0; i < m; i++)
    {
        System.out.print(hash[arr1[i]] + " ");
    }
}

// Driver code
public static void main (String[] args)
{
    int arr1[] = {1, 2, 3, 4, 7, 9};
    int arr2[] = {0, 1, 2, 1, 1, 4};
    int m, n;

    m = arr1.length;
    n = arr2.length;

    countEleLessThanOrEqual(arr1, m, arr2, n);
}
}

// This code is contributed
// by inder_verma
```

Output:

```
4 5 5 6 6 6
```

Improved By : inderDuMCA

Source

https://www.geeksforgeeks.org/for-each-element-in-1st-array-count-elements-less-than-or-equal-to-it-in-2nd-array-

Chapter 136

Frequency Measuring Techniques for Competitive Programming

Frequency Measuring Techniques for Competitive Programming - GeeksforGeeks

Measuring frequency of elements in an array is a really handy skill and is required a lot of competitive coding problems. We, in lot of problems are required to measure frequency of various elements like numbers, alphabets, symbols, etc. as a part of our problem.

Naive method

```
Input :  arr[] = {10, 20, 20, 10, 10, 20, 5, 20}
Output : 10 3
         20 4
         5  1

Input : arr[] = {10, 20, 20}
Output : 10 2
         20 1
```

We run two loops. For every item count number of times it occurs. To avoid duplicate printing, keep track of processed items.

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n)
{
```

```
    // Mark all array elements as not visited
    vector<int> visited(n, false);

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++) {

        // Skip this element if already processed
        if (visited[i] == true)
            continue;

        // Count frequency
        int count = 1;
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                visited[j] = true;
                count++;
            }
        }
        cout << arr[i] << " " << count << endl;
    }
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
10 3
20 4
5 1
```

Optimized methods :

Measuring frequencies when elements are limited by value
If our input array has small values, we can use array elements as index in a count array and increment count. In below example, elements are maximum 10.

```
Input :  arr[] = {5, 5, 6, 6, 5, 6, 1, 2, 3, 10, 10}
         limit = 10
Output : 1 1
```

```
2 1
3 1
5 3
6 3
10 2
```

```
 // CPP program to count frequencies of array items
// having small values.
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n, int limit)
{
    // Create an array to store counts. The size
    // of array is limit+1 and all values are
    // initially 0
    vector<int> count(limit+1, 0);

    // Traverse through array elements and
    // count frequencies (assuming that elements
    // are limited by limit)
    for (int i = 0; i < n; i++)
        count[arr[i]]++;

    for (int i = 0; i <= limit; i++)
       if (count[i] > 0)
        cout << i << " " << count[i] << endl;

}

int main()
{
    int arr[] = {5, 5, 6, 6, 5, 6, 1, 2, 3, 10, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    int limit = 10;
    countFreq(arr, n, limit);
    return 0;
}
```

Output:

```
1 1
2 1
3 1
5 3
6 3
10 2
```

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

const int limit = 255;

void countFreq(string str)
{
    // Create an array to store counts. The size
    // of array is limit+1 and all values are
    // initially 0
    vector<int> count(limit+1, 0);

    // Traverse through string characters and
    // count frequencies
    for (int i = 0; i < str.length(); i++)
        count[str[i]]++;

    for (int i = 0; i <= limit; i++)
       if (count[i] > 0)
        cout << (char)i << " " << count[i] << endl;

}

int main()
{
    string str = "GeeksforGeeks";
    countFreq(str);
    return 0;
}
```

Output:

```
G 2
e 4
f 1
k 2
o 1
r 1
s 2
```

Measuring frequencies when elements are in limited range
For example consider a string containing only upper case alphabets. Elements of string are limited in range from 'A' to 'Z'. The idea is to subtract smallest element ('A' in this example) to get index of the element.

```
 // CPP program to count frequencies of array items
```

```
#include <bits/stdc++.h>
using namespace std;

const int limit = 25;

void countFreq(string str)
{
    // Create an array to store counts. The size
    // of array is limit+1 and all values are
    // initially 0
    vector<int> count(limit+1, 0);

    // Traverse through string characters and
    // count frequencies
    for (int i = 0; i < str.length(); i++)
        count[str[i] - 'A']++;

    for (int i = 0; i <= limit; i++)
       if (count[i] > 0)
        cout << (char)(i + 'A') << " " << count[i] << endl;

}

int main()
{
    string str = "GEEKSFORGEEKS";
    countFreq(str);
    return 0;
}
```

Output:

```
E 4
F 1
G 2
K 2
O 1
R 1
S 2
```

Measuring frequencies if no range and no limit

The idea is to use hashing (unordered_map in C++ and HashMap in Java) to get frequencies.

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
```

```
using namespace std;

void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Traverse through map and print frequencies
    for (auto x : mp)
        cout << x.first << " " << x.second << endl;
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
5 1
10 3
20 4
```

Output:

```
5 1
10 3
20 4
```

Time Complexity : O(n)
Auxiliary Space : O(n)

In above efficient solution, how to print elements in same order as they appear in input?

```
 // CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;
```

```
void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // To print elements according to first
    // occurrence, traverse array one more time
    // print frequencies of elements and mark
    // frequencies as -1 so that same element
    // is not printed multiple times.
    for (int i = 0; i < n; i++) {
      if (mp[arr[i]] != -1)
      {
          cout << arr[i] << " " << mp[arr[i]] << endl;
          mp[arr[i]] = -1;
      }
    }
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
10 3
20 4
5 1
```

Output:

```
10 3
20 4
5 1
```

Time Complexity : O(n)
Auxiliary Space : O(n)

In Java, we can get elements in same order using LinkedHashMap. Therefore we do not need an extra loop.

Lot of problems are based on frequency measurement and will be a cheesecake if we know how to calculate frequency of various elements in a given array. For example try the given below problems which are based on frequency measurement:

1. **Anagrams**
2. **Sorting Elements of an Array by Frequency**
3. **Single Number**

Source

https://www.geeksforgeeks.org/frequency-measurement-techniques-for-competitive-programming/

Chapter 137

Frequency of a string in an array of strings

Frequency of a string in an array of strings - GeeksforGeeks

You are given a collection of strings and a list of queries. For every query there is a string given. We need to print the number of times the given string occurs in the collection of strings.
Examples:

```
Input : arr[] = {wer, wer, tyu, oio, tyu}
        q[] =   {wer, tyu, uio}
Output : 2 2 0
Explanation :
q[0] appears two times in arr[], q1[] appears
```

Method 1 (Simple)

The idea is simple, for every query string we compare it with all strings given in array. If the query string is matches, we increment count.

Java

```
 // Java program to find number of times a string
// appears in an array.
class SubString
{
    /* Returns count of occurrences of s in arr[] */
    static int search(String[]arr, String s)
    {
            int counter = 0;
```

```
            for (int j = 0; j < arr.length; j++)

                /* checking if string given in query is
                  present in the given string. If present,
                  increase times*/
                if (s.equals(arr[j]))
                    counter++;

            return counter;
    }

    static void answerQueries(String[] arr, String q[])
    {
        for (int i=0;i<q.length; i++)
            System.out.print(search(arr, q[i]) + " ");
    }

    /* driver code*/
    public static void main(String[] args) {

        String[] arr = {"aba","baba","aba","xzxb"};
        String[] q   = {"aba","xzxb","ab"};
        answerQueries(arr, q);
    }
}
```

C#

```
 // C# program to find number of
// times a string appears in an array.
using System;

class SubString
{
    /* Returns count of occurrences of s in arr[] */
    static int search(String[]arr, String s)
    {
            int counter = 0;
            for (int j = 0; j < arr.Length; j++)

                /* checking if string given in query is
                present in the given string. If present,
                increase times*/
                if (s.Equals(arr[j]))
                    counter++;

        return counter;
    }
```

```
    static void answerQueries(String []arr, String []q)
    {
        for (int i = 0; i < q.Length; i++)
            Console.Write(search(arr, q[i]) + " ");
    }

    // Driver code
    public static void Main()
    {
        String []arr = {"aba","baba","aba","xzxb"};
        String []q = {"aba","xzxb","ab"};
        answerQueries(arr, q);
    }
}

//This code is contributed by nitin mittal
```

Output:

```
2 1 0
```

Method 2 (Using Trie)

Trie an efficient data structure used for strong and retrieval of data like strings. The searching complexity is optimal as key length.
In this solution we insert the collection of strings in the Trie data structure so they get stored in it. One important thing is, we keep count of occurrences in leaf nodes. Then we search the Trie for the given query string and check if it is present in the Trie.

```
 // C++ program to count number of times
// a string appears in an array of strings
#include<iostream>
using namespace std;

const int MAX_CHAR = 26;

struct Trie
{
    // To store number of times
    // a string is present. It is
    // 0 is string is not present
    int cnt;

    Trie *node[MAX_CHAR];
```

```
    Trie()
    {
        for(int i=0; i<MAX_CHAR; i++)
            node[i] = NULL;
        cnt = 0;
    }
};

/* function to insert a string into the Trie*/
Trie *insert(Trie *root,string s)
{
    Trie *temp = root;
    int n = s.size();
    for(int i=0; i<n; i++)
    {
        /*calculation ascii value*/
        int index = s[i]-'a';

        /* If the given node is not already
          present in the Trie than create
          the new node*/
        if (!root->node[index])
            root->node[index] = new Trie();

        root = root->node[index];
    }
    root->cnt++;
    return temp;
}

/* Returns count of occurrences of s in Trie*/
int search(Trie *root, string s)
{
    int n = s.size();
    for (int i=0; i<n; i++)
    {
        int index = s[i]-'a';
        if (!root->node[index])
            return 0;
        root = root->node[index];
    }
    return root->cnt;
}

void answerQueries(string arr[], int n, string q[],
                                            int m)
{
    Trie *root = new Trie();
```

```
    /* inserting in Trie */
    for (int i=0; i<n; i++)
        root = insert(root,arr[i]);

    /* searching the strings in Trie */
    for (int i=0; i<m; i++)
        cout << search(root, q[i]) << " ";

}

/* Driver code */
int main()
{
    string arr[] = {"aba","baba","aba","xzxb"};
    int n = sizeof(arr)/sizeof(arr[0]);

    string q[] = {"aba","xzxb","ab"};
    int m = sizeof(q)/sizeof(q[0]);

    answerQueries(arr, n, q, m);

    return 0;
}
```

Output:

```
2
1
0
```

Method 3 (Hashing)

We can use a hash map and insert all given strings into it. For every query string, we simply do a look-up in the hash map.

Please referData Structure for Dictionary for comparison of hashing and Trie based solutions.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/frequency-of-a-string-in-an-array-of-strings/

Chapter 138

Game of replacing array elements

Game of replacing array elements - GeeksforGeeks

There are two players A and B who are interested in playing a game of numbers. In each move a player pick two distinct number, let's say *a1* and *a2* and then replace all *a2* by *a1* or *a1* by *a2*. They stop playing game if any one of them is unable to pick two number and the player who is unable to pick two distinct number in an array, looses the game. First player always move first and then second. Task is to find which player wins.

Examples:

```
Input :  arr[] = { 1, 3, 3, 2,, 2, 1 }
Output : Player 2 wins
Explanation:
First plays always looses irrespective
of the numbers chosen by him. For example,
say first player picks ( 1 & 3)
replace all 3 by 1
Now array Become { 1, 1, 1, 2, 2, 1 }
Then second player picks ( 1  2 )
either he replace 1 by 2 or 2 by 1
Array Become { 1, 1, 1, 1, 1, 1 }
Now first player is not able to choose.

Input  : arr[] = { 1, 2, 1, 2 }
Output : Player 1 wins
```

From above examples, we can observe that if number of count of distinct element is even, first player always wins. Else second player wins.

Lets take an another example :

```
  int arr[] =  1, 2, 3, 4, 5, 6
```

Here number of distinct element is even(n). If player 1 pick any two number lets say (4, 1), then we left with n-1 distinct element. So player second left with n-1 distinct element. This precess go on until distinct element become 1. Here $\mathbf{n = 6}$

```
Player   :  P1    p2    P1   p2    P1     P2
distinct : [n, n-1, n-2, n-3, n-4, n-5 ]

"At this point no distinct element left,
so p2 is unable to pick two Dis element."
```

Below c++ implementation of above idea :

```
 // CPP program for Game of Replacement
#include <bits/stdc++.h>
using namespace std;

// Function return which player win the game
int playGame(int arr[], int n)
{
    // Create hash that will stores
    // all distinct element
    unordered_set<int> hash;

    // Traverse an array element
    for (int i = 0; i < n; i++)
        hash.insert(arr[i]);

    return (hash.size() % 2 == 0 ? 1 : 2);
}

// Driver Function
int main()
{
    int arr[] = { 1, 1, 2, 2, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Player " << playGame(arr, n) << " Wins" << endl;
    return 0;
}
```

Output:

```
Player 1 Wins
```

Time Complexity : O(n)

Source

https://www.geeksforgeeks.org/game-replacing-array-elements/

Chapter 139

Given a sequence of words, print all anagrams together using STL

Given a sequence of words, print all anagrams together using STL - GeeksforGeeks

Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

Other approaches are discussed here in these posts
1) given-a-sequence-of-words-print-all-anagrams-together
2) given-a-sequence-of-words-print-all-anagrams-together-set-2

Here is HashMap solution using C++ Standard Template Library.
Approach :
1) Store the vector elements in HashMap with key as the sorted string
2) if key is same, then add string to value of HashMap(string vector)
3) Traverse the HashMap and print the anagram strings

```
 // CPP program for finding all anagram
// pairs in the given array
#include <iostream>
#include <algorithm>
#include <unordered_map>
#include <vector>
using namespace std;

// utility function for printing anagram list
void printAnagram(unordered_map<string,
                               vector<string> >& store)
{
    unordered_map<string, vector<string> >::iterator it;
```

```
    for (it = store.begin(); it != store.end(); it++) {
        vector<string> temp_vec(it->second);
        int size = temp_vec.size();
        if (size > 1) {
            for (int i = 0; i < size; i++) {
                cout << temp_vec[i] << " ";
            }
            cout << "\n";
        }
    }
}

// utility function for storing the vector of strings
// into HashMap
void storeInMap(vector<string>& vec)
{
    unordered_map<string, vector<string> > store;
    for (int i = 0; i < vec.size(); i++) {

        string tempString(vec[i]);
        sort(tempString.begin(), tempString.end());

        // check for sorted string if it already exists
        if (store.find(tempString) == store.end()) {
            vector<string> temp_vec;
            temp_vec.push_back(vec[i]);
            store.insert(make_pair(tempString, temp_vec));
        }

        else {
            // push new string to already existing key
            vector<string> temp_vec(store[tempString]);
            temp_vec.push_back(vec[i]);
            store[tempString] = temp_vec;
        }
    }

    // print utility function for printing
    // all the anagrams
    printAnagram(store);
}

// Driver code
int main()
{
    // initialize vector of strings
    vector<string> arr;
    arr.push_back("geeksquiz");
```

```
    arr.push_back("geeksforgeeks");
    arr.push_back("abcd");
    arr.push_back("forgeeksgeeks");
    arr.push_back("zuiqkeegs");
    arr.push_back("cat");
    arr.push_back("act");
    arr.push_back("tca");

    // utility function for storing strings
    // into hashmap
    storeInMap(arr);
    return 0;
}
```

Note : Compile above program with -std=c++11 flag in g++
Output:

```
cat act tca
geeksforgeeks forgeeksgeeks
geeksquiz zuiqkeegs
```

Source

https://www.geeksforgeeks.org/given-a-sequence-of-words-print-all-anagrams-together-using-stl/

Chapter 140

Given a sequence of words, print all anagrams together | Set 1

Given a sequence of words, print all anagrams together | Set 1 - GeeksforGeeks

Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

A **simple method** is to create a Hash Table. Calculate the hash value of each word in such a way that all anagrams have the same hash value. Populate the Hash Table with these hash values. Finally, print those words together with same hash values. A simple hashing mechanism can be modulo sum of all characters. With modulo sum, two non-anagram words may have same hash value. This can be handled by matching individual characters.

Following is **another method** to print all anagrams together. Take two auxiliary arrays, index array and word array. Populate the word array with the given sequence of words. Sort each individual word of the word array. Finally, sort the word array and keep track of the corresponding indices. After sorting, all the anagrams cluster together. Use the index array to print the strings from the original array of strings.

Let us understand the steps with following input Sequence of Words:

```
"cat", "dog", "tac", "god", "act"
```

1) Create two auxiliary arrays index[] and words[]. Copy all given words to words[] and store the original indexes in index[]

```
index[]:  0   1   2   3   4
words[]: cat dog tac god act
```

2) Sort individual words in words[]. Index array doesn’t change.

```
index[]:   0    1    2    3    4
words[]:  act  dgo  act  dgo  act
```

3) Sort the words array. Compare individual words using strcmp() to sort

```
index:     0    2    4    1    3
words[]:  act  act  act  dgo  dgo
```

4) All anagrams come together. But words are changed in words array. To print the original words, take index from the index array and use it in the original array. We get

```
"cat tac act dog god"
```

Following are the implementations of the above algorithm. In the following program, an array of structure "Word" is used to store both index and word arrays. DupArray is another structure that stores array of structure "Word".

C/C++

```
 // A C program to print all anagarms together
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// structure for each word of duplicate array
struct Word {
    char* str; // to store word itself
    int index; // index of the word in the original array
};

// structure to represent duplicate array.
struct DupArray {
    struct Word* array; // Array of words
    int size; // Size of array
};

// Create a DupArray object that contains an array of Words
struct DupArray* createDupArray(char* str[], int size)
{
    // Allocate memory for dupArray and all members of it
    struct DupArray* dupArray = (struct DupArray*)malloc(sizeof(struct DupArray));
    dupArray->size = size;
    dupArray->array = (struct Word*)malloc(dupArray->size * sizeof(struct Word));

    // One by one copy words from the given wordArray to dupArray
```

```
    int i;
    for (i = 0; i < size; ++i) {
        dupArray->array[i].index = i;
        dupArray->array[i].str = (char*)malloc(strlen(str[i]) + 1);
        strcpy(dupArray->array[i].str, str[i]);
    }

    return dupArray;
}

// Compare two characters. Used in qsort() for sorting an array of
// characters (Word)
int compChar(const void* a, const void* b)
{
    return *(char*)a - *(char*)b;
}

// Compare two words. Used in qsort() for sorting an array of words
int compStr(const void* a, const void* b)
{
    struct Word* a1 = (struct Word*)a;
    struct Word* b1 = (struct Word*)b;
    return strcmp(a1->str, b1->str);
}

// Given a list of words in wordArr[],
void printAnagramsTogether(char* wordArr[], int size)
{
    // Step 1: Create a copy of all words present in given wordArr.
    // The copy will also have orignal indexes of words
    struct DupArray* dupArray = createDupArray(wordArr, size);

    // Step 2: Iterate through all words in dupArray and sort
    // individual words.
    int i;
    for (i = 0; i < size; ++i)
        qsort(dupArray->array[i].str,
              strlen(dupArray->array[i].str), sizeof(char), compChar);

    // Step 3: Now sort the array of words in dupArray
    qsort(dupArray->array, size, sizeof(dupArray->array[0]), compStr);

    // Step 4: Now all words in dupArray are together, but these
    // words are changed. Use the index member of word struct to
    // get the corresponding original word
    for (i = 0; i < size; ++i)
        printf("%s ", wordArr[dupArray->array[i].index]);
}
```

```
// Driver program to test above functions
int main()
{
    char* wordArr[] = { "cat", "dog", "tac", "god", "act" };
    int size = sizeof(wordArr) / sizeof(wordArr[0]);
    printAnagramsTogether(wordArr, size);
    return 0;
}
```

Java

```
 // A Java program to print all anagrams together
import java.util.Arrays;
import java.util.Comparator;
public class GFG {
    // class for each word of duplicate array
    static class Word {
        String str; // to store word itself
        int index; // index of the word in the
        // original array

        // constructor
        Word(String str, int index)
        {
            this.str = str;
            this.index = index;
        }
    }

    // class to represent duplicate array.
    static class DupArray {
        Word[] array; // Array of words
        int size; // Size of array

        // constructor
        public DupArray(String str[], int size)
        {
            this.size = size;
            array = new Word[size];

            // One by one copy words from the
            // given wordArray to dupArray
            int i;
            for (i = 0; i < size; ++i) {
                // create a word Object with the
                // str[i] as str and index as i
                array[i] = new Word(str[i], i);
```

```
        }
    }
}

// Compare two words. Used in Arrays.sort() for
// sorting an array of words
static class compStr implements Comparator<Word> {
    public int compare(Word a, Word b)
    {
        return a.str.compareTo(b.str);
    }
}

// Given a list of words in wordArr[],
static void printAnagramsTogether(String wordArr[],
                                  int size)
{
    // Step 1: Create a copy of all words present
    // in given wordArr. The copy will also have
    // original indexes of words
    DupArray dupArray = new DupArray(wordArr, size);

    // Step 2: Iterate through all words in
    // dupArray and sort individual words.
    int i;
    for (i = 0; i < size; ++i) {
        char[] char_arr = dupArray.array[i].str.toCharArray();
        Arrays.sort(char_arr);
        dupArray.array[i].str = new String(char_arr);
    }

    // Step 3: Now sort the array of words in
    // dupArray
    Arrays.sort(dupArray.array, new compStr());

    // Step 4: Now all words in dupArray are together,
    // but these words are changed. Use the index
    // member of word struct to get the corresponding
    // original word
    for (i = 0; i < size; ++i)
        System.out.print(wordArr[dupArray.array[i].index] + " ");
}

// Driver program to test above functions
public static void main(String args[])
{
    String wordArr[] = { "cat", "dog", "tac", "god", "act" };
    int size = wordArr.length;
```

```
        printAnagramsTogether(wordArr, size);
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # A Python program to print all anagarms together

# structure for each word of duplicate array
class Word(object):
    def __init__(self, string, index):
        self.string = string
        self.index = index

# Create a DupArray object that contains an array
# of Words
def createDupArray(string, size):
    dupArray = []

    # One by one copy words from the given wordArray
    # to dupArray
    for i in xrange(size):
        dupArray.append(Word(string[i], i))

    return dupArray

# Given a list of words in wordArr[]
def printAnagramsTogether(wordArr, size):
    # Step 1: Create a copy of all words present in
    # given wordArr.
    # The copy will also have orignal indexes of words
    dupArray = createDupArray(wordArr, size)

    # Step 2: Iterate through all words in dupArray and sort
    # individual words.
    for i in xrange(size):
        dupArray[i].string = ''.join(sorted(dupArray[i].string))

    # Step 3: Now sort the array of words in dupArray
    dupArray = sorted(dupArray, key = lambda k: k.string)

    # Step 4: Now all words in dupArray are together, but
    # these words are changed. Use the index member of word
    # struct to get the corresponding original word
    for word in dupArray:
        print wordArr[word.index],
```

```
# Driver program
wordArr = ["cat", "dog", "tac", "god", "act"]
size = len(wordArr)
printAnagramsTogether(wordArr, size)

# This code is contributed by BHAVYA JAIN
```

Output:

```
cat tac act dog god
```

Time Complexity: Let there be N words and each word has maximum M characters. The upper bound is O(NMLogM + MNLogN).
Step 2 takes O(NMLogM) time. Sorting a word takes maximum O(MLogM) time. So sorting N words takes O(NMLogM) time. step 3 takes O(MNLogN) Sorting array of words takes NLogN comparisons. A comparison may take maximum O(M) time. So time to sort array of words will be O(MNLogN).

Using hashmap

Here, we will sort each word, calculate its hashcode and then put it in a map where the key will be hashcode generated after sorting. The value of the map will be a list containing all the words which have same hashcode after sorting.
Lastly, we will print all values from the hashmap where size of values will be greater than 1.

```
 import java.util.*;

public class FindAnagrams {

    private static void printAnagrams(String arr[])
    {
        HashMap<Integer, List<String> > map = new HashMap<>();

        // loop over all words
        for (int i = 0; i < arr.length; i++) {

            // convert to char array, sort and
            // then re-convert to string
            String word = arr[i];
            char[] letters = word.toCharArray();
            Arrays.sort(letters);
            String newWord = new String(letters);

            // calculate hashcode of string
            // after sorting
            int n = newWord.hashCode();
            if (map.containsKey(n)) {
```

```
                // Here, we already have
                // a word for the hashcode
                List<String> words = map.get(n);
                words.add(word);
                map.put(n, words);
            }
            else {

                // This is the first time we are
                // adding a word for a specific
                // hashcode
                List<String> words = new ArrayList<>();
                words.add(word);
                map.put(n, words);
            }
        }

        // print all the values where size is > 1
        // If you want to print non-anagrams,
        // just print the values having size = 1
        for (Integer i : map.keySet()) {
            List<String> values = map.get(i);
            if (values.size() > 1) {
                System.out.print(values);
            }
        }
    }

    public static void main(String[] args)
    {

        // Driver program
        String arr[] = { "cat", "dog", "tac", "god", "act" };
        printAnagrams(arr);
    }
}
```

Output :

```
[cat, tac, act][dog, god]
```

Given a sequence of words, print all anagrams together | Set 2

Improved By : subharaj01

Source

https://www.geeksforgeeks.org/given-a-sequence-of-words-print-all-anagrams-together/

Chapter 141

Given a sequence of words, print all anagrams together | Set 2

Given a sequence of words, print all anagrams together | Set 2 - GeeksforGeeks

Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

We have discussed two different methods in the previous post. In this post, a more efficient solution is discussed.

Trie data structure can be used for a more efficient solution. Insert the sorted order of each word in the trie. Since all the anagrams will end at the same leaf node. We can start a linked list at the leaf nodes where each node represents the index of the original array of words. Finally, traverse the Trie. While traversing the Trie, traverse each linked list one line at a time. Following are the detailed steps.

1) Create an empty Trie
2) One by one take all words of input sequence. Do following for each word
…**a)** Copy the word to a buffer.
…**b)** Sort the buffer
…**c)** Insert the sorted buffer and index of this word to Trie. Each leaf node of Trie is head of a Index list. The Index list stores index of words in original sequence. If sorted buffe is already present, we insert index of this word to the index list.
3) Traverse Trie. While traversing, if you reach a leaf node, traverse the index list. And print all words using the index obtained from Index list.

C++

```
 // An efficient program to print all anagrams together
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <ctype.h>

#define NO_OF_CHARS 26

// Structure to represent list node for indexes of words in
// the given sequence. The list nodes are used to connect
// anagrams at leaf nodes of Trie
struct IndexNode
{
    int index;
    struct IndexNode* next;
};

// Structure to represent a Trie Node
struct TrieNode
{
    bool isEnd;  // indicates end of word
    struct TrieNode* child[NO_OF_CHARS]; // 26 slots each for 'a' to 'z'
    struct IndexNode* head; // head of the index list
};


// A utility function to create a new Trie node
struct TrieNode* newTrieNode()
{
    struct TrieNode* temp = new TrieNode;
    temp->isEnd = 0;
    temp->head = NULL;
    for (int i = 0; i < NO_OF_CHARS; ++i)
        temp->child[i] = NULL;
    return temp;
}

/* Following function is needed for library function qsort(). Refer
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare(const void* a, const void* b)
{  return *(char*)a - *(char*)b; }

/* A utility function to create a new linked list node */
struct IndexNode* newIndexNode(int index)
{
    struct IndexNode* temp = new IndexNode;
    temp->index = index;
    temp->next = NULL;
    return temp;
}

// A utility function to insert a word to Trie
```

```
void insert(struct TrieNode** root, char* word, int index)
{
    // Base case
    if (*root == NULL)
        *root = newTrieNode();

    if (*word != '\0')
        insert( &( (*root)->child[tolower(*word) - 'a'] ), word+1, index );
    else  // If end of the word reached
    {
        // Insert index of this word to end of index linked list
        if ((*root)->isEnd)
        {
            IndexNode* pCrawl = (*root)->head;
            while( pCrawl->next )
                pCrawl = pCrawl->next;
            pCrawl->next = newIndexNode(index);
        }
        else  // If Index list is empty
        {
            (*root)->isEnd = 1;
            (*root)->head = newIndexNode(index);
        }
    }
}

// This function traverses the built trie. When a leaf node is reached,
// all words connected at that leaf node are anagrams. So it traverses
// the list at leaf node and uses stored index to print original words
void printAnagramsUtil(struct TrieNode* root, char *wordArr[])
{
    if (root == NULL)
        return;

    // If a lead node is reached, print all anagrams using the indexes
    // stored in index linked list
    if (root->isEnd)
    {
        // traverse the list
        IndexNode* pCrawl = root->head;
        while (pCrawl != NULL)
        {
            printf( "%s ", wordArr[ pCrawl->index ] );
            pCrawl = pCrawl->next;
        }
    }

    for (int i = 0; i < NO_OF_CHARS; ++i)
```

```
        printAnagramsUtil(root->child[i], wordArr);
}

// The main function that prints all anagrams together. wordArr[] is input
// sequence of words.
void printAnagramsTogether(char* wordArr[], int size)
{
    // Create an empty Trie
    struct TrieNode* root = NULL;

    // Iterate through all input words
    for (int i = 0; i < size; ++i)
    {
        // Create a buffer for this word and copy the word to buffer
        int len = strlen(wordArr[i]);
        char *buffer = new char[len+1];
        strcpy(buffer, wordArr[i]);

        // Sort the buffer
        qsort( (void*)buffer, strlen(buffer), sizeof(char), compare );

        // Insert the sorted buffer and its original index to Trie
        insert(&root,  buffer, i);
    }

    // Traverse the built Trie and print all anagrms together
    printAnagramsUtil(root, wordArr);
}


// Driver program to test above functions
int main()
{
    char* wordArr[] = {"cat", "dog", "tac", "god", "act", "gdo"};
    int size = sizeof(wordArr) / sizeof(wordArr[0]);
    printAnagramsTogether(wordArr, size);
    return 0;
}
```

Java

```
 // An efficient program to print all
// anagrams together
import java.util.Arrays;
import java.util.LinkedList;

public class GFG
{
```

```
static final int NO_OF_CHARS = 26;

// Class to represent a Trie Node
static class TrieNode
{
    boolean isEnd;  // indicates end of word

    // 26 slots each for 'a' to 'z'
    TrieNode[] child = new TrieNode[NO_OF_CHARS];

    // head of the index list
    LinkedList<Integer> head;

    // constructor
    public TrieNode()
    {
        isEnd = false;
        head = new LinkedList<>();
        for (int i = 0; i < NO_OF_CHARS; ++i)
            child[i] = null;
    }
}

// A utility function to insert a word to Trie
static TrieNode insert(TrieNode root,String word,
                           int index, int i)
{
    // Base case
    if (root == null)
    {
        root = new TrieNode();
    }

    if (i < word.length() )
    {
        int index1 = word.charAt(i) - 'a';
        root.child[index1] = insert(root.child[index1],
                                    word, index, i+1 );
    }
    else  // If end of the word reached
    {
        // Insert index of this word to end of
        // index linked list
        if (root.isEnd == true)
        {
            root.head.add(index);
        }
        else // If Index list is empty
```

```
        {
            root.isEnd = true;
            root.head.add(index);
        }
    }
    return root;
}

// This function traverses the built trie. When a leaf
// node is reached, all words connected at that leaf
// node are anagrams. So it traverses the list at leaf
// node and uses stored index to print original words
static void printAnagramsUtil(TrieNode root,
                                  String wordArr[])
{
    if (root == null)
        return;

    // If a lead node is reached, print all anagrams
    // using the indexes stored in index linked list
    if (root.isEnd)
    {
        // traverse the list
        for(Integer pCrawl: root.head)
            System.out.println(wordArr[pCrawl]);
    }

    for (int i = 0; i < NO_OF_CHARS; ++i)
        printAnagramsUtil(root.child[i], wordArr);
}

// The main function that prints all anagrams together.
// wordArr[] is input sequence of words.
static void printAnagramsTogether(String wordArr[],
                                          int size)
{
    // Create an empty Trie
    TrieNode root = null;

    // Iterate through all input words
    for (int i = 0; i < size; ++i)
    {
        // Create a buffer for this word and copy the
        // word to buffer
        char[] buffer = wordArr[i].toCharArray();

        // Sort the buffer
        Arrays.sort(buffer);
```

```
            // Insert the sorted buffer and its original
            // index to Trie
            root = insert(root, new String(buffer), i, 0);

        }

        // Traverse the built Trie and print all anagrms
        // together
        printAnagramsUtil(root, wordArr);
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        String wordArr[] = {"cat", "dog", "tac", "god",
                                        "act", "gdo"};
        int size = wordArr.length;
        printAnagramsTogether(wordArr, size);
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
cat
tac
act
dog
god
gdo
```

Improved By : reyaz_ahmed

Source

https://www.geeksforgeeks.org/given-a-sequence-of-words-print-all-anagrams-together-set-2/

Chapter 142

Given an array A[] and a number x, check for pair in A[] with sum as x

Given an array A[] and a number x, check for pair in A[] with sum as x - GeeksforGeeks

Write a program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.

METHOD 1 (Use Sorting)

Algorithm :

```
hasArrayTwoCandidates (A[], ar_size, sum)
1) Sort the array in non-decreasing order.
2) Initialize two index variables to find the candidate
   elements in the sorted array.
       (a) Initialize first to the leftmost index: l = 0
       (b) Initialize second  the rightmost index:  r = ar_size-1
3) Loop while l < r.
       (a) If (A[l] + A[r] == sum)  then return 1
       (b) Else if( A[l] + A[r] <  sum )  then l++
       (c) Else r--
4) No candidates in whole array - return 0
```

Time Complexity: Depends on what sorting algorithm we use. If we use Merge Sort or Heap Sort then (-)(nlogn) in worst case. If we use Quick Sort then O(n^2) in worst case.
Auxiliary Space : Again, depends on sorting algorithm. For example auxiliary space is O(n) for merge sort and O(1) for Heap Sort.

Example :
Let Array be {1, 4, 45, 6, 10, -8} and sum to find be 16

Sort the array
A = {-8, 1, 4, 6, 10, 45}

Initialize l = 0, r = 5
A[l] + A[r] (-8 + 45) > 16 => decrement r. Now r = 10
A[l] + A[r] (-8 + 10) increment l. Now l = 1
A[l] + A[r] (1 + 10) increment l. Now l = 2
A[l] + A[r] (4 + 10) increment l. Now l = 3
A[l] + A[r] (6 + 10) == 16 => Found candidates (return 1)

Note: If there are more than one pair having the given sum then this algorithm reports only one. Can be easily extended for this though.

Below is the implementation of the above approach.

C

```
 // C program to check if given array
// has 2 elements whose sum is equal
// to the given value

# include <stdio.h>
# define bool int

void quickSort(int *, int, int);

bool hasArrayTwoCandidates(int A[], int arr_size, int sum)
{
    int l, r;

    /* Sort the elements */
    quickSort(A, 0, arr_size-1);

    /* Now look for the two candidates in the sorted
       array*/
    l = 0;
    r = arr_size-1;
    while (l < r)
    {
         if(A[l] + A[r] == sum)
              return 1;
         else if(A[l] + A[r] < sum)
              l++;
         else // A[i] + A[j] > sum
              r--;
    }
    return 0;
```

```
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING
    PURPOSE */
void exchange(int *a, int *b)
{
    int temp;
    temp = *a;
    *a   = *b;
    *b   = temp;
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++)
    {
        if(A[j] <= x)
        {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange (&A[i + 1], &A[ei]);
    return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
void quickSort(int A[], int si, int ei)
{
    int pi;    /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/* Driver program to test above function */
int main()
```

```
{
    int A[] = {1, 4, 45, 6, 10, -8};
    int n = 16;
    int arr_size = 6;

    if( hasArrayTwoCandidates(A, arr_size, n))
        printf("Array has two elements with given sum");
    else
        printf("Array doesn't have two elements with given sum");

    getchar();
    return 0;
}
```

C++

```
 // C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value

#include <bits/stdc++.h>

using namespace std;

// Function to check if array has 2 elements
// whose sum is equal to the given value
bool hasArrayTwoCandidates(int A[], int arr_size,
                                          int sum)
{
    int l, r;

    /* Sort the elements */
    sort(A, A + arr_size);

    /* Now look for the two candidates in
       the sorted array*/
    l = 0;
    r = arr_size - 1;
    while (l < r)
    {
        if(A[l] + A[r] == sum)
            return 1;
        else if(A[l] + A[r] < sum)
            l++;
        else // A[i] + A[j] > sum
            r--;
    }
    return 0;
```

```
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, -8};
    int n = 16;
    int arr_size = sizeof(A) / sizeof(A[0]);

    // Function calling
    if(hasArrayTwoCandidates(A, arr_size, n))
        cout << "Array has two elements with given sum";
    else
        cout << "Array doesn't have two elements with given sum";

    return 0;
}
```

Java

```
 // Java program to check if given array
// has 2 elements whose sum is equal
// to the given value
import java.util.*;

class GFG
{
    // Function to check if array has 2 elements
    // whose sum is equal to the given value
    static boolean hasArrayTwoCandidates(int A[],
                            int arr_size, int sum)
    {
        int l, r;

        /* Sort the elements */
        Arrays.sort(A);

        /* Now look for the two candidates
        in the sorted array*/
        l = 0;
        r = arr_size-1;
        while (l < r)
        {
            if(A[l] + A[r] == sum)
                return true;
            else if(A[l] + A[r] < sum)
                l++;
            else // A[i] + A[j] > sum
```

```
                r--;
        }
        return false;
    }

    // Driver code
    public static void main(String args[])
    {
        int A[] = {1, 4, 45, 6, 10, -8};
        int n = 16;
        int arr_size = A.length;

        // Function calling
        if(hasArrayTwoCandidates(A, arr_size, n))
            System.out.println("Array has two " +
                               "elements with given sum");
        else
            System.out.println("Array doesn't have " +
                               "two elements with given sum");

    }

}
```

Python

```
 # Python program to check for the sum condition to be satisified

def hasArrayTwoCandidates(A,arr_size,sum):

    # sort the array
    quickSort(A,0,arr_size-1)
    l = 0
    r = arr_size-1

    # traverse the array for the two elements
    while l<r:
        if (A[l] + A[r] == sum):
            return 1
        elif (A[l] + A[r] < sum):
            l += 1
        else:
            r -= 1
    return 0

# Implementation of Quick Sort
# A[] --> Array to be sorted
# si  --> Starting index
```

```
# ei  --> Ending index
def quickSort(A, si, ei):
    if si < ei:
        pi=partition(A,si,ei)
        quickSort(A,si,pi-1)
        quickSort(A,pi+1,ei)

# Utility function for partitioning the array(used in quick sort)
def partition(A, si, ei):
    x = A[ei]
    i = (si-1)
    for j in range(si,ei):
        if A[j] <= x:
            i += 1

            # This operation is used to swap two variables is python
            A[i], A[j] = A[j], A[i]

        A[i+1], A[ei] = A[ei], A[i+1]

    return i+1


# Driver program to test the functions
A = [1,4,45,6,10,-8]
n = 16
if (hasArrayTwoCandidates(A, len(A), n)):
    print("Array has two elements with the given sum")
else:
    print("Array doesn't have two elements with the given sum")

## This code is contributed by __Devesh Agrawal__
```

C#

```
 // C# program to check for pair
// in A[] with sum as x

using System;

class GFG
{
    static bool hasArrayTwoCandidates(int []A,
                                int arr_size, int sum)
    {
        int l, r;

        /* Sort the elements */
```

```
    sort(A, 0, arr_size-1);

    /* Now look for the two candidates
    in the sorted array*/
    l = 0;
    r = arr_size-1;
    while (l < r)
    {
        if(A[l] + A[r] == sum)
            return true;
        else if(A[l] + A[r] < sum)
            l++;
        else // A[i] + A[j] > sum
            r--;
    }
    return false;
}

/* Below functions are only to sort the
array using QuickSort */

/* This function takes last element as pivot,
places the pivot element at its correct
position in sorted array, and places all
smaller (smaller than pivot) to left of
pivot and all greater elements to right
of pivot */
static int partition(int []arr, int low, int high)
{
    int pivot = arr[high];

    // index of smaller element
    int i = (low-1);
    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller
        // than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
```

```
        // swap arr[i+1] and arr[high] (or pivot)
        int temp1 = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp1;

        return i+1;
    }

    /* The main function that
    implements QuickSort()
    arr[] --> Array to be sorted,
    low --> Starting index,
    high --> Ending index */
    static void sort(int []arr, int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi]
            is now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    // Driver code
    public static void Main()
    {
        int []A = {1, 4, 45, 6, 10, -8};
        int n = 16;
        int arr_size = 6;

        if( hasArrayTwoCandidates(A, arr_size, n))
            Console.Write("Array has two elements"+
                        " with given sum");
        else
            Console.Write("Array doesn't have "+
                       "two elements with given sum");

    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to check if given
// array has 2 elements whose sum
// is equal to the given value

// Function to check if array has
// 2 elements whose sum is equal
// to the given value
function hasArrayTwoCandidates($A, $arr_size,
                                        $sum)
{
    $l; $r;

    /* Sort the elements */
    //sort($A, A + arr_size);
    sort($A);

    /* Now look for the two candidates
    in the sorted array*/
    $l = 0;
    $r = $arr_size - 1;
    while ($l < $r)
    {
        if($A[$l] + $A[$r] == $sum)
            return 1;
        else if($A[$l] + $A[$r] < $sum)
            $l++;
        else // A[i] + A[j] > sum
            $r--;
    }
    return 0;
}

// Driver Code
$A = array (1, 4, 45, 6, 10, -8);
$n = 16;
$arr_size = sizeof($A);

// Function calling
if(hasArrayTwoCandidates($A, $arr_size, $n))
    echo "Array has two elements " .
                    "with given sum";
else
    echo "Array doesn't have two " .
          "elements with given sum";

// This code is contributed by m_kit
?>
```

Output :

```
Array has two elements with the given sum
```

METHOD 2 (Use Hashing)
This method works in O(n) time.

```
1) Initialize an empty hash table s.
2) Do following for each element A[i] in A[]
   (a)    If s[x - A[i]] is set then print the pair (A[i], x - A[i])
   (b)    Insert A[i] into s.
```

Below is the implementation of the above approach :

C

```
 // C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value

// Works only if range elements is limited
#include <stdio.h>
#define MAX 100000

void printPairs(int arr[], int arr_size, int sum)
{
  int i, temp;
  bool s[MAX] = {0}; /*initialize hash set as 0*/

  for (i = 0; i < arr_size; i++)
  {
      temp = sum - arr[i];
      if (temp >= 0 && s[temp] == 1)
         printf("Pair with given sum %d is (%d, %d) n",
                  sum, arr[i], temp);
      s[arr[i]] = 1;
  }
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, 8};
    int n = 16;
    int arr_size = sizeof(A)/sizeof(A[0]);
```

```
    printPairs(A, arr_size, n);

    getchar();
    return 0;
}
```

C++

```
 // C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value
#include <bits/stdc++.h>

using namespace std;

void printPairs(int arr[], int arr_size, int sum)
{
    unordered_set<int> s;
    for (int i = 0; i < arr_size; i++)
    {
        int temp = sum - arr[i];

        if (temp >= 0 && s.find(temp) != s.end())
            cout << "Pair with given sum " << sum <<
                 " is (" << arr[i] << ", " << temp <<
                 ")" << endl;

        s.insert(arr[i]);
    }
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, 8};
    int n = 16;
    int arr_size = sizeof(A)/sizeof(A[0]);

    // Function calling
    printPairs(A, arr_size, n);

    return 0;
}
```

Java

```
 // Java implementation using Hashing
```

```
import java.io.*;
import java.util.HashSet;

class PairSum
{
    static void printpairs(int arr[],int sum)
    {
        HashSet<Integer> s = new HashSet<Integer>();
        for (int i=0; i<arr.length; ++i)
        {
            int temp = sum-arr[i];

            // checking for condition
            if (temp>=0 && s.contains(temp))
            {
                System.out.println("Pair with given sum " +
                                   sum + " is (" + arr[i] +
                                   ", "+temp+")");
            }
            s.add(arr[i]);
        }
    }

    // Main to test the above function
    public static void main (String[] args)
    {
        int A[] = {1, 4, 45, 6, 10, 8};
        int n = 16;
        printpairs(A,  n);
    }
}

// This article is contributed by Aakash Hasija
```

Python

```
 # Python program to find if there are
# two elements wtih given sum

# function to check for the given sum
# in the array
def printPairs(arr, arr_size, sum):

    # Create an empty hash set
    s = set()

    for i in range(0,arr_size):
        temp = sum-arr[i]
```

```
        if (temp>=0 and temp in s):
            print ("Pair with the given sum is", arr[i], "and", temp)
        s.add(arr[i])

# driver program to check the above function
A = [1,4,45,6,10,8]
n = 16
printPairs(A, len(A), n)

# This code is contributed by __Devesh Agrawal__
```

C#

```
 // C# implementation using Hashing
using System;
using System.Collections.Generic;

class GFG
{
static void printpairs(int []arr,
                       int sum)
{
    HashSet<int> s = new HashSet<int>();
    for (int i = 0; i < arr.Length; ++i)
    {
        int temp = sum - arr[i];

        // checking for condition
        if (temp >= 0 && s.Contains(temp))
        {
            Console.Write("Pair with given sum " +
                          sum + " is (" + arr[i] +
                               ", " + temp + ")");
        }
        s.Add(arr[i]);
    }
}

// Driver Code
static void Main ()
{
    int []A = new int[]{1, 4, 45,
                        6, 10, 8};
    int n = 16;
    printpairs(A, n);
}
}
```

```
// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
Pair with given sum 16 is (10, 6)
```

Time Complexity: O(n)
Auxiliary Space: O(n) where n is size of array.

If range of numbers include negative numbers then also it works. All we have to do for negative numbers is to make everything positive by adding the absolute value of smallest negative integer to all numbers.

Related Problems:
Given two unsorted arrays, find all pairs whose sum is x
Count pairs with given sumCount all distinct pairs with difference equal to k

Improved By : jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/

Chapter 143

Given an array of pairs, find all symmetric pairs in it

Given an array of pairs, find all symmetric pairs in it - GeeksforGeeks

Two pairs (a, b) and (c, d) are said to be symmetric if c is equal to b and a is equal to d. For example (10, 20) and (20, 10) are symmetric. Given an array of pairs find all symmetric pairs in it.

It may be assumed that first elements of all pairs are distinct.

Example:

```
Input: arr[] = {{11, 20}, {30, 40}, {5, 10}, {40, 30}, {10, 5}}
Output: Following pairs have symmetric pairs
        (30, 40)
        (5, 10)
```

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is to go through every pair, and check every other pair for symmetric. This solution requires $O(n^2)$ time.

A **Better Solution** is to use sorting. Sort all pairs by first element. For every pair, do binary search for second element in the given array, i.e., check if second element of this pair exists as first element in array. If found, then compare first element of pair with second element. Time Complexity of this solution is O(nLogn).

An **Efficient Solution** is to use Hashing. First element of pair is used as key and second element is used as value. The idea is traverse all pairs one by one. For every pair, check if its second element is in hash table. If yes, then compare the first element with value of matched entry of hash table. If the value and the first element match, then we found symmetric pairs. Else, insert first element as key and second element as value.

Following are C++ and Java implementation of this idea.

C/C++

```
 #include<bits/stdc++.h>
using namespace std;

// A C++ program to find all symmetric pairs in a given array of pairs
// Print all pairs that have a symmetric counterpart
void findSymPairs(int arr[][2], int row)
{
    // Creates an empty hashMap hM
    unordered_map<int, int> hM;

    // Traverse through the given array
    for (int i = 0; i < row; i++)
    {
        // First and second elements of current pair
        int first = arr[i][0];
        int sec   = arr[i][1];

        // If found and value in hash matches with first
        // element of this pair, we found symmetry
        if (hM.find(sec) != hM.end() && hM[sec] == first)
            cout << "(" << sec << ", " << first << ")" <<endl;

        else  // Else put sec element of this pair in hash
            hM[first] = sec;
    }
}

// Drive method
int main()
{
    int arr[5][2];
    arr[0][0] = 11; arr[0][1] = 20;
    arr[1][0] = 30; arr[1][1] = 40;
    arr[2][0] = 5;  arr[2][1] = 10;
    arr[3][0] = 40;  arr[3][1] = 30;
    arr[4][0] = 10;  arr[4][1] = 5;
    findSymPairs(arr, 5);
}

//This is contributed by Chhavi
```

Java

```
 // A Java program to find all symmetric pairs in a given array of pairs
```

```
import java.util.HashMap;

class SymmetricPairs {

    // Print all pairs that have a symmetric counterpart
    static void findSymPairs(int arr[][])
    {
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        // Traverse through the given array
        for (int i = 0; i < arr.length; i++)
        {
            // First and second elements of current pair
            int first = arr[i][0];
            int sec   = arr[i][1];

            // Look for second element of this pair in hash
            Integer val = hM.get(sec);

            // If found and value in hash matches with first
            // element of this pair, we found symmetry
            if (val != null && val == first)
               System.out.println("(" + sec + ", " + first + ")");

            else  // Else put sec element of this pair in hash
               hM.put(first, sec);
        }
    }

    // Drive method
    public static void main(String arg[])
    {
        int arr[][] = new int[5][2];
        arr[0][0] = 11; arr[0][1] = 20;
        arr[1][0] = 30; arr[1][1] = 40;
        arr[2][0] = 5;  arr[2][1] = 10;
        arr[3][0] = 40;  arr[3][1] = 30;
        arr[4][0] = 10;  arr[4][1] = 5;
        findSymPairs(arr);
    }
}
```

Output:

```
Following pairs have symmetric pairs
(30, 40)
(5, 10)
```

Time Complexity of this solution is O(n) under the assumption that hash search and insert methods work in O(1) time.

This article is contributed by **Shivam Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/given-an-array-of-pairs-find-all-symmetric-pairs-in-it/

Chapter 144

Given two unsorted arrays, find all pairs whose sum is x

Given two unsorted arrays, find all pairs whose sum is x - GeeksforGeeks

Given two unsorted arrays of distinct elements, the task is to find all pairs from both arrays whose sum is equal to x.

Examples:

```
Input :  arr1[] = {-1, -2, 4, -6, 5, 7}
         arr2[] = {6, 3, 4, 0}
         x = 8
Output : 4 4
         5 3

Input : arr1[] = {1, 2, 4, 5, 7}
        arr2[] = {5, 6, 3, 4, 8}
        x = 9
Output : 1 8
         4 5
         5 4
```

Asked in : Amazon

A **Naive approach** is to simply run two loops and pick elements from both arrays. One by one check that both elements sum is equal to given value x or not.

C++

```
 // C++ program to find all pairs in both arrays
// whose sum is equal to given value x
```

```
#include<bits/stdc++.h>

using namespace std;

// Function to print all pairs in both arrays
// whose sum is equal to given value x
void findPairs(int arr1[], int arr2[], int n,
                               int m, int x)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (arr1[i] + arr2[j] == x)
                cout << arr1[i] << " "
                     << arr2[j] << endl;
}

// Driver code
int main()
{
    int arr1[] = {1, 2, 3, 7, 5, 4};
    int arr2[] = {0, 7, 4, 3, 2, 1};
    int n = sizeof(arr1)/sizeof(int);
    int m = sizeof(arr2)/sizeof(int);
    int x = 8;
    findPairs(arr1, arr2, n, m, x);
    return 0;
}
```

Java

```
 // Java program to find all pairs in both arrays
// whose sum is equal to given value x
import java.io.*;

class GFG
{

    // Function to print all pairs in both arrays
    // whose sum is equal to given value x
    static void findPairs(int arr1[], int arr2[], int n,
                                          int m, int x)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (arr1[i] + arr2[j] == x)
                    System.out.println( arr1[i] + " "
                                          + arr2[j] );
    }
```

```
    // Driver code
    public static void main(String[] args)
    {
        int arr1[] = {1, 2, 3, 7, 5, 4};
        int arr2[] = {0, 7, 4, 3, 2, 1};
        int x = 8;
        findPairs(arr1, arr2, arr1.length, arr2.length, x);
    }

}

// This code is contributed
// by sunnysingh
```

Python3

```
 # Python 3 program to find all
# pairs in both arrays whose
# sum is equal to given value x

# Function to print all pairs
# in both arrays whose sum is
# equal to given value x
def findPairs(arr1, arr2, n, m, x):

    for i in range(0, n):
        for j in range(0, m):
            if (arr1[i] + arr2[j] == x):
                print(arr1[i], arr2[j])

# Driver code
arr1 = [1, 2, 3, 7, 5, 4]
arr2 = [0, 7, 4, 3, 2, 1]
n = len(arr1)
m = len(arr2)
x = 8
findPairs(arr1, arr2, n, m, x)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find all
// pairs in both arrays
// whose sum is equal to
// given value x
```

```
using System;

class GFG
{

// Function to print all
// pairs in both arrays
// whose sum is equal to
// given value x
static void findPairs(int []arr1, int []arr2,
                      int n, int m, int x)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (arr1[i] + arr2[j] == x)
                Console.WriteLine(arr1[i] + " " +
                                  arr2[j] );
}

// Driver code
static void Main()
{
    int []arr1 = {1, 2, 3, 7, 5, 4};
    int []arr2 = {0, 7, 4, 3, 2, 1};
    int x = 8;
    findPairs(arr1, arr2,
              arr1.Length,
              arr2.Length, x);
}
}

// This code is contributed
// by Sam007
```

PHP

```
 <?php
// PHP program to find all pairs
// in both arrays whose sum is
// equal to given value x

// Function to print all pairs
// in both arrays whose sum is
// equal to given value x
function findPairs($arr1, $arr2,
                   $n, $m, $x)
{
    for ($i = 0; $i < $n; $i++)
```

```
        for ($j = 0; $j < $m; $j++)
            if ($arr1[$i] + $arr2[$j] == $x)
                echo $arr1[$i] . " " .
                     $arr2[$j] . "\n";
}

// Driver code
$arr1 = array(1, 2, 3, 7, 5, 4);
$arr2 = array(0, 7, 4, 3, 2, 1);
$n = count($arr1);
$m = count($arr2);
$x = 8;
findPairs($arr1, $arr2,
          $n, $m, $x);

// This code is contributed
// by Sam007
?>
```

Output:

```
1 7
7 1
5 3
4 4
```

Time Complexity : O(n^2)
Auxiliary Space : O(1)

An **Efficient solution** of this problem is to hashing. Hash table is implemented using unordered_set in C++. We store all first array elements in hash table. For elements of second array, we subtract every element from x and check the result in hash table. If result is present, we print the element and key in hash (which is an element of first array).

C++

```
 // C++ program to find all pair in both arrays
// whose sum is equal to given value x
#include<bits/stdc++.h>
using namespace std;

// Function to find all pairs in both arrays
// whose sum is equal to given value x
void findPairs(int arr1[], int arr2[], int n,
                               int m, int x)
{
```

```
    // Insert all elements of first array in a hash
    unordered_set<int> s;
    for (int i = 0; i < n; i++)
        s.insert(arr1[i]);

    // Subtract sum from second array elements one
    // by one and check it's present in array first
    // or not
    for (int j = 0; j < m; j++)
        if (s.find(x - arr2[j]) != s.end())
            cout << x-arr2[j] << " "
                 << arr2[j] << endl;
}

// Driver code
int main()
{
    int arr1[] = {1, 0, -4, 7, 6, 4};
    int arr2[] = {0 ,2, 4, -3, 2, 1};
    int x = 8;
    int n = sizeof(arr1)/sizeof(int);
    int m = sizeof(arr2)/sizeof(int);
    findPairs(arr1, arr2, n, m, x);
    return 0;
}
```

Java

```
 // JAVA Code for Given two unsorted arrays,
// find all pairs whose sum is x
import java.util.*;

class GFG {

     // Function to find all pairs in both arrays
    // whose sum is equal to given value x
    public static void findPairs(int arr1[], int arr2[],
                                 int n, int m, int x)
    {
        // Insert all elements of first array in a hash
        HashMap<Integer, Integer> s = new HashMap<Integer, Integer>();

        for (int i = 0; i < n; i ++)
            s.put(arr1[i], 0);

        // Subtract sum from second array elements one
        // by one and check it's present in array first
        // or not
```

```
        for (int j = 0; j < m; j ++)
            if (s.containsKey(x - arr2[j]))
                System.out.println(x - arr2[j] + " " + arr2[j]);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr1[] = {1, 0, -4, 7, 6, 4};
        int arr2[] = {0 ,2, 4, -3, 2, 1};
        int x = 8;

        findPairs(arr1, arr2, arr1.length, arr2.length, x);

    }
  }
// This code is contributed by Arnav Kr. Mandal.
```

Output:

```
6 2
4 4
6 2
7 1
```

Time Complexity : O(nlog(n))
Auxiliary Space : O(n)

Improved By : Sam007, namankatuva

Source

https://www.geeksforgeeks.org/given-two-unsorted-arrays-find-pairs-whose-sum-x/

Chapter 145

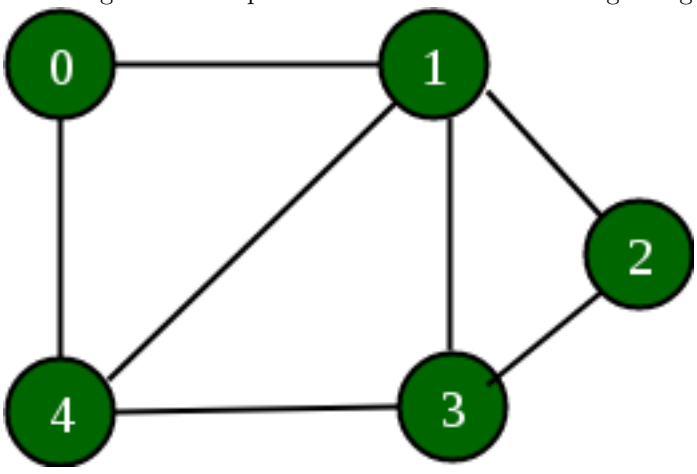
Graph representations using set and hash

Graph representations using set and hash - GeeksforGeeks

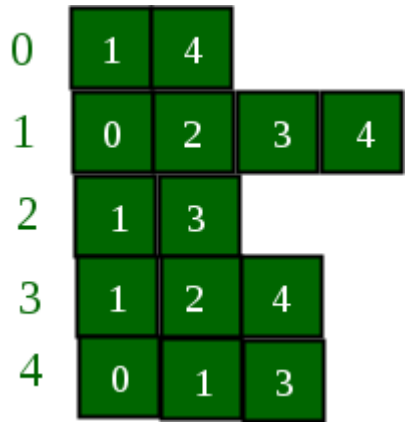
We have introduced Graph implementation using array of vectors in Graph implementation using STL for competitive programming | Set 1. In this post, a different implementation is used which can be used to implement graphs using sets. The implementation is for adjacency list representation of graph.

A set is different from a vector in two ways: it stores elements in a sorted way, and duplicate elements are not allowed. Therefore, this approach cannot be used for graphs containing parallel edges. Since sets are internally implemented as binary search trees, **an edge between two vertices can be searched in O(logV) time**, where V is the number of vertices in the graph.

Following is an example of an undirected and unweighted graph with 5 vertices.



Below is adjacency list representation of this graph using array of sets.



Below is the code for adjacency list representation of an undirected graph using sets:

```
 // A C++ program to demonstrate adjacency list
// representation of graphs using sets
#include <bits/stdc++.h>
using namespace std;

struct Graph {
    int V;
    set<int, greater<int> >* adjList;
};

// A utility function that creates a graph of V vertices
Graph* createGraph(int V)
{
    Graph* graph = new Graph;
    graph->V = V;

    // Create an array of sets representing
    // adjacency lists.  Size of the array will be V
    graph->adjList = new set<int, greater<int> >[V];

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(Graph* graph, int src, int dest)
{
    // Add an edge from src to dest.  A new
    // element is inserted to the adjacent
    // list of src.
    graph->adjList[src].insert(dest);

    // Since graph is undirected, add an edge
```

```
    // from dest to src also
    graph->adjList[dest].insert(src);
}

// A utility function to print the adjacency
// list representation of graph
void printGraph(Graph* graph)
{
    for (int i = 0; i < graph->V; ++i) {
        set<int, greater<int> > lst = graph->adjList[i];
        cout << endl << "Adjacency list of vertex "
             << i << endl;

        for (auto itr = lst.begin(); itr != lst.end(); ++itr)
            cout << *itr << " ";
        cout << endl;
    }
}

// Searches for a given edge in the graph
void searchEdge(Graph* graph, int src, int dest)
{
    auto itr = graph->adjList[src].find(dest);
    if (itr == graph->adjList[src].end())
        cout << endl << "Edge from " << src
             << " to " << dest << " not found."
             << endl;
    else
        cout << endl << "Edge from " << src
             << " to " << dest << " found."
             << endl;
}

// Driver code
int main()
{
    // Create the graph given in the above figure
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    // Print the adjacency list representation of
```

```
    // the above graph
    printGraph(graph);

    // Search the given edge in the graph
    searchEdge(graph, 2, 1);
    searchEdge(graph, 0, 3);

    return 0;
}
```

Output:

```
Adjacency list of vertex 0
4 1

Adjacency list of vertex 1
4 3 2 0

Adjacency list of vertex 2
3 1

Adjacency list of vertex 3
4 2 1

Adjacency list of vertex 4
3 1 0

Edge from 2 to 1 found.

Edge from 0 to 3 not found.
```

Pros: Queries like whether there is an edge from vertex u to vertex v can be done in O(log V).

Cons:

- Adding an edge takes O(log V), as opposed to O(1) in vector implementation.
- Graphs containing parallel edge(s) cannot be implemented through this method.

Further Optimization of Edge Search Operation using unordered_set (or hashing):
The edge search operation can be further optimized to O(1) using unordered_set which uses hashing internally.

```
 // A C++ program to demonstrate adjacency list
// representation of graphs using sets
```

```
#include <bits/stdc++.h>
using namespace std;

struct Graph {
    int V;
    unordered_set<int>* adjList;
};

// A utility function that creates a graph of
// V vertices
Graph* createGraph(int V)
{
    Graph* graph = new Graph;
    graph->V = V;

    // Create an array of sets representing
    // adjacency lists. Size of the array will be V
    graph->adjList = new unordered_set<int>[V];

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(Graph* graph, int src, int dest)
{
    // Add an edge from src to dest. A new
    // element is inserted to the adjacent
    // list of src.
    graph->adjList[src].insert(dest);

    // Since graph is undirected, add an edge
    // from dest to src also
    graph->adjList[dest].insert(src);
}

// A utility function to print the adjacency
// list representation of graph
void printGraph(Graph* graph)
{
    for (int i = 0; i < graph->V; ++i) {
        unordered_set<int> lst = graph->adjList[i];
        cout << endl << "Adjacency list of vertex "
             << i << endl;

        for (auto itr = lst.begin(); itr != lst.end(); ++itr)
            cout << *itr << " ";
        cout << endl;
    }
```

```
}

// Searches for a given edge in the graph
void searchEdge(Graph* graph, int src, int dest)
{
    auto itr = graph->adjList[src].find(dest);
    if (itr == graph->adjList[src].end())
        cout << endl << "Edge from " << src
             << " to " << dest << " not found."
             << endl;
    else
        cout << endl << "Edge from " << src
             << " to " << dest << " found."
             << endl;
}

// Driver code
int main()
{
    // Create the graph given in the above figure
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    // Print the adjacency list representation of
    // the above graph
    printGraph(graph);

    // Search the given edge in the graph
    searchEdge(graph, 2, 1);
    searchEdge(graph, 0, 3);

    return 0;
}
```

Output :

```
Adjacency list of vertex 0
4 1

Adjacency list of vertex 1
```

```
4 3 2 0

Adjacency list of vertex 2
3 1

Adjacency list of vertex 3
4 2 1

Adjacency list of vertex 4
3 1 0

Edge from 2 to 1 found.

Edge from 0 to 3 not found.
```

Pros:

- Queries like whether there is an edge from vertex u to vertex v can be done in O(1).
- Adding an edge takes O(1).

Cons:

- Graphs containing parallel edge(s) cannot be implemented through this method.
- Edges are not stored in any order.

Note : adjacency matrix representation is the most optimized for edge search, but space requirements of adjacency matrix are comparatively high for big sparse graphs. Moreover adjacency matrix has other disadvantages as well like BFS and DFS become costly as we can't quickly get all adjacent of a node.

Source

https://www.geeksforgeeks.org/graph-representations-using-set-hash/

Chapter 146

Group Shifted String

Group Shifted String - GeeksforGeeks

Given an array of strings (all lowercase letters), the task is to group them in such a way that all strings in a group are shifted versions of each other. Two string S and T are called shifted if,

```
S.length = T.length
and
S[i] = T[i] + K for
1 <= i <= S.length  for a constant integer K
```

For example strings {acd, dfg, wyz, yab, mop} are shifted versions of each other.

```
Input  : str[] = {"acd", "dfg", "wyz", "yab", "mop",
                  "bdfh", "a", "x", "moqs"};

Output : a x
         acd dfg wyz yab mop
         bdfh moqs
All shifted strings are grouped together.
```

We can see a pattern among string of one group, the difference between consecutive characters for all character of string are equal. As in above example take acd, dfg and mop

a c d -> 2 1
d f g -> 2 1
m o p -> 2 1

Since the differences are same, we can use this to identify strings that belong to same group. The idea is to form a string of differences as key. If a string with same difference string is

found, then this string also belongs to same group. For example, above three strings have same difference string, that is "21".

In below implementation, we add 'a' to every difference and store 21 as "ba".

```
 /* C/C++ program to print groups of shifted strings
   together. */
#include <bits/stdc++.h>
using namespace std;
const int ALPHA = 26;   // Total lowercase letter

// Method to a difference string for a given string.
// If string is "adf" then difference string will be
// "cb" (first difference 3 then difference 2)
string getDiffString(string str)
{
    string shift = "";
    for (int i = 1; i < str.length(); i++)
    {
        int dif = str[i] - str[i-1];
        if (dif < 0)
            dif += ALPHA;

        // Representing the difference as char
        shift += (dif + 'a');
    }

    // This string will be 1 less length than str
    return shift;
}

// Method for grouping shifted string
void groupShiftedString(string str[], int n)
{
    // map for storing indices of string which are
    // in same group
    map< string, vector<int> > groupMap;
    for (int i = 0; i < n; i++)
    {
        string diffStr = getDiffString(str[i]);
        groupMap[diffStr].push_back(i);
    }

    // iterating through map to print group
    for (auto it=groupMap.begin(); it!=groupMap.end();
                                                it++)
    {
        vector<int> v = it->second;
        for (int i = 0; i < v.size(); i++)
```

```
            cout << str[v[i]] << " ";
        cout << endl;
    }
}

// Driver method to test above methods
int main()
{
    string str[] = {"acd", "dfg", "wyz", "yab", "mop",
                    "bdfh", "a", "x", "moqs"
                   };
    int n = sizeof(str)/sizeof(str[0]);
    groupShiftedString(str, n);
    return 0;
}
```

Output:

```
a x
acd dfg wyz yab mop
bdfh moqs
```

Source

https://www.geeksforgeeks.org/group-shifted-string/

Chapter 147

Group multiple occurrence of array elements ordered by first occurrence

Group multiple occurrence of array elements ordered by first occurrence - GeeksforGeeks

Given an unsorted array with repetitions, the task is to group multiple occurrence of individual elements. The grouping should happen in a way that the order of first occurrences of all elements is maintained.

Examples:

```
Input: arr[] = {5, 3, 5, 1, 3, 3}
Output:        {5, 5, 3, 3, 3, 1}

Input: arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4}
Output:        {4, 4, 4, 6, 6, 9, 9, 2, 3, 10}
```

Simple Solution is to use nested loops. The outer loop traverses array elements one by one. The inner loop checks if this is first occurrence, if yes, then the inner loop prints it and all other occurrences.

C++

```
 // A simple C++ program to group multiple occurrences of individual
// array elements
#include<iostream>
using namespace std;

// A simple method to group all occurrences of individual elements
```

```
void groupElements(int arr[], int n)
{
    // Initialize all elements as not visited
    bool *visited = new bool[n];
    for (int i=0; i<n; i++)
        visited[i] = false;

    // Traverse all elements
    for (int i=0; i<n; i++)
    {
        // Check if this is first occurrence
        if (!visited[i])
        {
            // If yes, print it and all subsequent occurrences
            cout << arr[i] << " ";
            for (int j=i+1; j<n; j++)
            {
                if (arr[i] == arr[j])
                {
                    cout << arr[i] << " ";
                    visited[j] = true;
                }
            }
        }
    }

    delete [] visited;
}

/* Driver program to test above function */
int main()
{
    int arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    groupElements(arr, n);
    return 0;
}
```

Python3

```
 # A simple Python 3 program to
# group multiple occurrences of
# individual array elements

# A simple method to group all
# occurrences of individual elements
def groupElements(arr, n):
```

```
    # Initialize all elements
    # as not visited
    visited = [False] * n
    for i in range(0, n):
        visited[i] = False

    # Traverse all elements
    for i in range(0, n):

        # Check if this is
        # first occurrence
        if (visited[i] == False):

            # If yes, print it and
            # all subsequent occurrences
            print(arr[i], end = " ")
            for j in range(i + 1, n):

                if (arr[i] == arr[j]):

                    print(arr[i], end = " ")
                    visited[j] = True

# Driver Code
arr = [4, 6, 9, 2, 3,
       4, 9, 6, 10, 4]
n = len(arr)
groupElements(arr, n)

# This code is contributed
# by Smitha
```

Output:

```
4 4 4 6 6 9 9 2 3 10
```

Time complexity of the above method is $O(n^2)$.

Binary Search Tree based Method: The time complexity can be improved to O(nLogn) using self-balancing binary search tree like Red-Black Tree or AVL tree. Following is complete algorithm.
1) Create an empty Binary Search Tree (BST). Every BST node is going to contain an array element and its count.
2) Traverse the input array and do following for every element.
…….a) If element is not present in BST, then insert it with count as 0.
…….b) If element is present, then increment count in corresponding BST node.
3) Traverse the array again and do following for every element.
……. If element is present in BST, then do following

……..a) Get its count and print the element ‘count’ times.
……..b) Delete the element from BST.

Time Complexity of the above solution is O(nLogn).

Hashing based Method: We can also use hashing. The idea is to replace Binary Search Tree with a Hash Map in above algorithm.

Below is Implementation of hashing based solution.

Java

```
 /* Java program to group multiple occurrences of individual array elements */
import java.util.HashMap;

class Main
{
    // A hashing based method to group all occurrences of individual elements
    static void orderedGroup(int arr[])
    {
        // Creates an empty hashmap
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        // Traverse the array elements, and store count for every element
        // in HashMap
        for (int i=0; i<arr.length; i++)
        {
           // Check if element is already in HashMap
           Integer prevCount = hM.get(arr[i]);
           if (prevCount == null)
                prevCount = 0;

           // Increment count of element element in HashMap
           hM.put(arr[i], prevCount + 1);
        }

        // Traverse array again
        for (int i=0; i<arr.length; i++)
        {
            // Check if this is first occurrence
            Integer count =  hM.get(arr[i]);
            if (count != null)
            {
                // If yes, then print the element 'count' times
                for (int j=0; j<count; j++)
                   System.out.print(arr[i] + " ");

                // And remove the element from HashMap.
                hM.remove(arr[i]);
```

```
            }
        }
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int arr[] = {10, 5, 3, 10, 10, 4, 1, 3};
        orderedGroup(arr);
    }
}
```

Output:

```
10 10 10 5 3 3 4 1
```

Time Complexity of the above hashing based solution is $\Theta(n)$ under the assumption that insert, search and delete operations on HashMap take O(1) time.

Below is a related problem for strings.
Group all occurrences of characters according to first appearance

This article is contributed by **Himanshu Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/group-multiple-occurrence-of-array-elements-ordered-by-first-occurrence/

Chapter 148

Group words with same set of characters

Group words with same set of characters - GeeksforGeeks

Given a list of words with lower cases. Implement a function to find all Words that have the same unique character set .

Example:

```
Input: words[] = { "may", "student", "students", "dog",
                  "studentssess", "god", "cat", "act",
                  "tab", "bat", "flow", "wolf", "lambs",
                  "amy", "yam", "balms", "looped",
                  "poodle"};
Output :
looped, poodle,
lambs, balms,
flow, wolf,
tab, bat,
may, amy, yam,
student, students, studentssess,
dog, god,
cat, act,

All words with same set of characters are printed
together in a line.
```

The idea is to use hashing. We generate a key for all words. The key contains all unique character (Size of key is at most 26 for lower case alphabets). We store indexes of words as values for a key. Once we have filled all keys and values in hash table, we can print the result by traversing the table.

Below is the implementation of above idea .

C++

```
 // C++ program to print all words that have
// the same unique character set
#include<bits/stdc++.h>
using namespace std;
#define MAX_CHAR 26

// Generates a key from given string. The key
// contains all unique characters of given string
// in sorted order.
string getKey(string &str)
{
    bool visited[MAX_CHAR] = { false };

    // store all unique characters of current
    // word in key
    for (int j = 0; j < str.length(); j++)
        visited[str[j] - 'a'] = true ;
    string key = "";
    for (int j=0; j < MAX_CHAR; j++)
        if (visited[j])
            key = key + (char)('a'+j);
    return key;
}

// Print all words together with same character sets.
void wordsWithSameCharSet(string words[], int n)
{
    // Stores indexes of all words that have same
    // set of unique characters.
    unordered_map <string, vector <int> > Hash;

    // Traverse all words
    for (int i=0; i<n; i++)
    {
        string key = getKey(words[i]);
        Hash[key].push_back(i);
    }

    // print all words that have the same unique character set
    for (auto it = Hash.begin(); it!=Hash.end(); it++)
    {
      for (auto v=(*it).second.begin(); v!=(*it).second.end(); v++)
          cout << words[*v] << ", ";
      cout << endl;
```

```
    }
}

// Driver program to test above function
int main()
{
    string words[] = { "may", "student", "students", "dog",
                 "studentssess", "god", "cat", "act", "tab",
                 "bat", "flow", "wolf", "lambs", "amy", "yam",
                 "balms", "looped", "poodle"};
    int n = sizeof(words)/sizeof(words[0]);
    wordsWithSameCharSet(words, n);
    return 0;
}
```

Java

```
 // Java program to print all words that have
// the same unique character set
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map.Entry;
public class GFG {

    static final int MAX_CHAR = 26;

    // Generates a key from given string. The key
    // contains all unique characters of given string
    // in sorted order.
    static String getKey(String str)
    {
        boolean[] visited = new boolean[MAX_CHAR];
        Arrays.fill(visited, false);

        // store all unique characters of current
        // word in key
        for (int j = 0; j < str.length(); j++)
            visited[str.charAt(j) - 'a'] = true ;
        String key = "";
        for (int j=0; j < MAX_CHAR; j++)
            if (visited[j])
                key = key + (char)('a'+j);
        return key;
    }

    // Print all words together with same character sets.
    static void wordsWithSameCharSet(String words[], int n)
```

```
{
    // Stores indexes of all words that have same
    // set of unique characters.
    //unordered_map <string, vector <int> > Hash;
    HashMap<String, ArrayList<Integer>> Hash = new HashMap<>();

    // Traverse all words
    for (int i=0; i<n; i++)
    {
        String key = getKey(words[i]);

        // if the key is already in the map
        // then get its corresponding value
        // and update the list and put it in the map
        if(Hash.containsKey(key))
        {
            ArrayList<Integer> get_al = Hash.get(key);
            get_al.add(i);
            Hash.put(key, get_al);
        }

        // if key is not present in the map
        // then create a new list and add
        // both key and the list
        else
        {
            ArrayList<Integer> new_al = new ArrayList<>();
            new_al.add(i);
            Hash.put(key, new_al);
        }
    }

    // print all words that have the same unique character set
    for (Entry<String, ArrayList<Integer>> it : Hash.entrySet())
    {
        ArrayList<Integer> get =it.getValue();
        for (Integer v:get)
            System.out.print( words[v] + ", ");
        System.out.println();
    }
}

// Driver program to test above function
public static void main(String args[])
{
    String words[] = { "may", "student", "students", "dog",
                "studentssess", "god", "cat", "act", "tab",
                "bat", "flow", "wolf", "lambs", "amy", "yam",
```

```
                    "balms", "looped", "poodle"};
        int n = words.length;
        wordsWithSameCharSet(words, n);
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Function to group all strings with same characters
from collections import Counter

def groupStrings(input):
    # traverse all strings one by one
    # dict is an empty dictionary
    dict={}

    for word in input:
        # sort the current string and take it's
        # sorted value as key
        # sorted return list of sorted characters
        # we need to join them to get key as string
        # Counter() method returns dictionary with frequency of
        # each character as value
        wordDict=Counter(word)

        # now get list of keys
        key = wordDict.keys()

        # now sort these keys
        key = sorted(key)

        # join these characters to produce key string
        key = ''.join(key)

        # now check if this key already exist in
        # dictionary or not
        # if exist then simply append current word
        # in mapped list on key
        # otherwise first assign empty list to key and
        # then append current word in it
        if key in dict.keys():
            dict[key].append(word)
        else:
            dict[key]=[]
            dict[key].append(word)

        # now traverse complete dictionary and print
```

```
        # list of mapped strings in each key seprated by ,
    for (key,value) in dict.iteritems():
        print ','.join(dict[key])

# Driver program
if __name__ == "__main__":
    input=['may','student','students','dog','studentssess','god','cat','act','tab','bat','flow',
    groupStrings(input)
```

Output:

```
looped, poodle,
lambs, balms,
flow, wolf,
tab, bat,
may, amy, yam,
student, students, studentssess,
dog, god,
cat, act,
```

Time complexity : O(n*k) where n is number of words in dictionary and k is maximum length of a word.

Source

https://www.geeksforgeeks.org/print-words-together-set-characters/

Chapter 149

Hash Table vs STL Map

Hash Table vs STL Map - GeeksforGeeks

This article focus on : Compare and contrast Hash table and an STL Map. How is the hash table implemented? If the number of inputs is small, which data structure options can be used instead of a hash table?

Hash table

In a hash table, a value is stored by calling a hash function on a key.

- Values are not stored in sorted order.
- Additionally, since hash tables use the key to find the index that will store the value, an insert or lookup can be done in amortised O(1) time (assuming few collisions in the hash table).
- In a hash table, one must also handle potential collisions. This is often done by chaining, which means to create a linked list of all the values whose keys map to a particular index.

Implementation of Hash Table : A hash table is traditionally implemented with an array of linked lists. When we want to insert a key/Value pair, we map the key to an index in the array using the hash function. The value is then inserted into the linked list at that position.
Note: The elements in the linked list at a particular index of the array do not have the same key. Rather, hash function(key) is the same for these values. Therefore, in order to retrieve the value for a specific key,we need to store in each node both the exact key and the value.

To summarize, a hash table will be implemented with an array of linked lists, where each node in the linked list holds two pieces of data: the value and the original key. In addition, we will want to note the following design criteria:

1. We want to use a good hash function to ensure that the keys are well distributed. If they are not well distributed, then we would get a lot of collision and the speed to find an element would decline.

2. No matter how good hash function is, we will still have collisions, so we need a method for handling them. this often means chaining via a linked list, but it's not the only way.
3. We may also wish to implement methods to dynamically increase or decrease the hash table size depending on capacity. For example, when the ratio of the number of elements to the table size exceeds a certain threshold, we may wish to increase the hash table size. This would mean creating a new hash table and transferring the entries from the old table to the new table. Because this is an expensive operation, we want to be careful to not do it too often.

STL Map

The container map is an associative container included in the standard library of C++. The definition of this class is in the header file "map" of the namespace std.

STL Map Internal Implementation:
It's implemented as a self-balancing red-black tree. Probably the two most common self balancing trees are red-black tree and AVL trees. To balance the tree after an insertion/update both algorithms use the notion of rotations where the nodes of the tree are rotated to perform the re-balancing. While in both algorithms the insert/delete operations are O(log n), in the case of Red-Black tree re-balancing rotation is an O(1) operation while with AVL this is a O(log n) operation, making the RB tree more efficient in this aspect of the re-balancing sage and one of the possible reasons that is more commonly used.

Differences between hash table and STL map

1. **Null Keys :** STL Map allows one null key and multiple null values whereas hash table doesn't allow any null key or value.
2. **Thread synchronization :** Map is generally preferred over hash table if thread synchronization is not needed. Hash table is synchronized.
3. **Thread safe:** STL Maps are not thread safe whereas Hashmaps are thread safe and can be shared with many threads.
4. **Value Order :** In STL map, values are stored in sorted order whereas in hash table values are not stored in sorted order
5. **Searching Time :** You can use STL Map or binary tree for smaller data(Although it takes O(log n) time, the number of inputs may be small enough to make this time negligible) and for large amount of data, hash table is preferred.

Related articles

- Advantages of BST over hashmap
- Differences between HashMap and HashTable in Java

Source

https://www.geeksforgeeks.org/hash-table-vs-stl-map/

Chapter 150

HashSet vs TreeSet in Java

HashSet vs TreeSet in Java - GeeksforGeeks

- **Speed and internal implementation**
  HashSet : For operations like search, insert and delete. It takes constant time for these operations on average. HashSet is faster than TreeSet. HashSet is Implemented using a hash table.

  TreeSet : TreeSet takes O(Log n) for search, insert and delete which is higher than HashSet. But TreeSet keeps sorted data. Also, it supports operations like higher() (Returns least higher element), floor(), ceiling(), etc. These operations are also O(Log n) in TreeSet and not supported in HashSet. TreeSet is implemented using a Self Balancing Binary Search Tree (Red-Black Tree). TreeSet is backed by TreeMap in Java.

- **Ordering**
  Elements in HashSet are not ordered. TreeSet maintains objects in Sorted order defined by either Comparable or Comparator method in Java. TreeSet elements are sorted in ascending order by default. It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.

- **Null Object**
  HashSet allows null object. TreeSet doesn't allow null Object and throw NullPointerException, Why, because TreeSet uses compareTo() method to compare keys and compareTo() will throw java.lang.NullPointerException.

- **Comparison**
  HashSet uses equals() method to compare two object in Set and for detecting duplicates. TreeSet uses compareTo() method for same purpose.
  If equals() and compareTo() are not consistent, i.e. for two equal object equals should return true while compareTo() should return zero, than it will break contract of Set interface and will allow duplicates in Set implementations like TreeSet

If you want a sorted Set then it is better to add elements to HashSet and then convert it into TreeSet rather than creating a TreeSet and adding elements to it.

HashSet example

```
 // Java program to demonstrate working of
// HashSet
import java.util.HashSet;
class HashSetDemo {
    public static void main(String[] args)
    {

        // Create a HashSet
        HashSet<String> hset = new HashSet<String>();

        // add elements to HashSet
        hset.add("geeks");
        hset.add("for");
        hset.add("practice");
        hset.add("contribute");

        // Duplicate removed
        hset.add("geeks");

        // Displaying HashSet elements
        System.out.println("HashSet contains: ");
        for (String temp : hset) {
            System.out.println(temp);
        }
    }
}
```

Output:

```
HashSet contains:
practice
geeks
for
contribute
```

TreeSet example

```
 // Java program to demonstrate working of
// TreeSet.
import java.util.TreeSet;
class TreeSetDemo {

    public static void main(String[] args)
    {
```

```
        // Create a TreeSet
        TreeSet<String> tset = new TreeSet<String>();

        // add elements to HashSet
        tset.add("geeks");
        tset.add("for");
        tset.add("practice");
        tset.add("contribute");

        // Duplicate removed
        tset.add("geeks");

        // Displaying TreeSet elements
        System.out.println("TreeSet contains: ");
        for (String temp : tset) {
            System.out.println(temp);
        }
    }
}
```

Output:

```
TreeSet contains:
contribute
for
geeks
practice
```

When to prefer TreeSet over HashSet

1. Sorted unique elements are required instead of unique elements. The sorted list given by TreeSet is always in ascending order.

2. TreeSet has greater locality than HashSet.If two entries are near by in the order, then TreeSet places them near each other in data structure and hence in memory, while HashSet spreads the entries all over memory regardless of the keys they are associated to.

3. TreeSet uses Red- Black tree algorithm underneath to sort out the elements. When one need to perform read/write operations frequently, then TreeSet is a good choice.

4. LinkedHashSet is another data structure that is between these two. It provides time complexities like HashSet and maintains order of insertion (Note that this is not sorted order, but the order in which elements are inserted).

Source

https://www.geeksforgeeks.org/hashset-vs-treeset-in-java/

Chapter 151

Hashing in Java

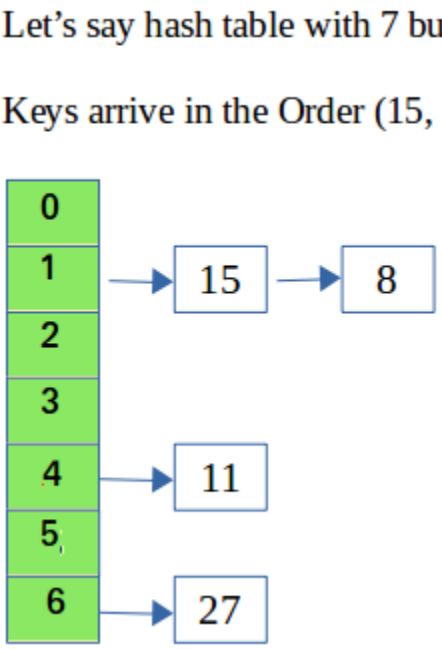
Hashing in Java - GeeksforGeeks

In hashing there is a hash function that maps keys to some values. But these hashing function may lead to collision that is two or more keys are mapped to same value. **Chain hashing** avoids collision. The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Let's create a hash function, such that our hash table has 'N' number of buckets.
To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.
Example: hashIndex = key % noOfBuckets

Insert: Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list.

Delete: To delete a node from hash table, calculate the hash index for the key, move to the bucket corresponds to the calculated hash index, search the list in the current bucket to find and remove the node with the given key (if found).



Please refer **Hashing | Set 2 (Separate Chaining)** for details.

1. With help of HashTable (A synchronized implementation of hashing)
2. With the help of HashMap (A non-synchronized faster implementation of hashing)

```
 // Java program to create HashMap from an array
// by taking the elements as Keys and
// the frequencies as the Values

import java.util.*;

class GFG {

    // Function to create HashMap from array
    static void createHashMap(int arr[])
    {
        // Creates an empty HashMap
        HashMap<Integer, Integer> hmap = new HashMap<Integer, Integer>();

        // Traverse through the given array
        for (int i = 0; i < arr.length; i++) {

            // Get if the element is present
            Integer c = hmap.get(arr[i]);
```

```
            // If this is first occurrence of element
            // Insert the element
            if (hmap.get(arr[i]) == null) {
                hmap.put(arr[i], 1);
            }

            // If elements already exists in hash map
            // Increment the count of element by 1
            else {
                hmap.put(arr[i], ++c);
            }
        }

        // Print HashMap
        System.out.println(hmap);
    }

    // Driver method to test above method
    public static void main(String[] args)
    {
        int arr[] = { 10, 34, 5, 10, 3, 5, 10 };
        createHashMap(arr);
    }
}
```

Output:

```
{34=1, 3=1, 5=2, 10=3}
```

3. With the help of LinkedHashMap (Similar to HashMap, but keeps order of elements)

```
 // Java program to demonstrate working of LinkedHashMap
import java.util.*;

public class BasicLinkedHashMap
{
    public static void main(String a[])
    {
        LinkedHashMap<String, String> lhm =
                        new LinkedHashMap<String, String>();
        lhm.put("one", "practice.geeksforgeeks.org");
        lhm.put("two", "code.geeksforgeeks.org");
        lhm.put("four", "quiz.geeksforgeeks.org");

        // It prints the elements in same order
        // as they were inserted
```

```
        System.out.println(lhm);

        System.out.println("Getting value for key 'one': "
                                        + lhm.get("one"));
        System.out.println("Size of the map: " + lhm.size());
        System.out.println("Is map empty? " + lhm.isEmpty());
        System.out.println("Contains key 'two'? "+
                                   lhm.containsKey("two"));
        System.out.println("Contains value 'practice.geeks"
        +"forgeeks.org'? "+ lhm.containsValue("practice"+
        ".geeksforgeeks.org"));
        System.out.println("delete element 'one': " +
                           lhm.remove("one"));
        System.out.println(lhm);
    }
}
```

Output:

```
{one=practice.geeksforgeeks.org, two=code.geeksforgeeks.org, four=quiz.geeksforgeeks.org}
Getting value for key 'one': practice.geeksforgeeks.org
Size of the map: 3
Is map empty? false
Contains key 'two'? true
Contains value 'practice.geeksforgeeks.org'? true
delete element 'one': practice.geeksforgeeks.org
{two=code.geeksforgeeks.org, four=quiz.geeksforgeeks.org}
```

4. With the help of ConcurretHashMap(Similar to Hashtable, Synchronized, but faster as multiple locks are used)

```
 // Java program to demonstrate working of ConcurrentHashMap

import java.util.concurrent.*;
class ConcurrentHashMapDemo {
    public static void main(String[] args)
    {
        ConcurrentHashMap<Integer, String> m =
                   new ConcurrentHashMap<Integer, String>();
        m.put(100, "Hello");
        m.put(101, "Geeks");
        m.put(102, "Geeks");

        // Printing the ConcurrentHashMap
        System.out.println("ConcurentHashMap: " + m);

        // Adding Hello at 101 key
```

```
        // This is already present in ConcurrentHashMap object
        // Therefore its better to use putIfAbsent for such cases
        m.putIfAbsent(101, "Hello");

        // Printing the ConcurrentHashMap
        System.out.println("\nConcurentHashMap: " + m);

        // Trying to remove entry for 101 key
        // since it is present
        m.remove(101, "Geeks");

        // Printing the ConcurrentHashMap
        System.out.println("\nConcurentHashMap: " + m);

        // replacing the value for key 101
        // from "Hello" to "For"
        m.replace(100, "Hello", "For");

        // Printing the ConcurrentHashMap
        System.out.println("\nConcurentHashMap: " + m);
    }
}
```

Output:

```
ConcurentHashMap: {100=Hello, 101=Geeks, 102=Geeks}

ConcurentHashMap: {100=Hello, 101=Geeks, 102=Geeks}

ConcurentHashMap: {100=Hello, 102=Geeks}

ConcurentHashMap: {100=For, 102=Geeks}
```

5. With the help of HashSet (Similar to HashMap, but maintains only keys, not pair)

```
 // Java program to demonstrate working of HashSet
import java.util.*;

class Test {
    public static void main(String[] args)
    {
        HashSet<String> h = new HashSet<String>();

        // Adding elements into HashSet usind add()
        h.add("India");
```

```
        h.add("Australia");
        h.add("South Africa");
        h.add("India"); // adding duplicate elements

        // Displaying the HashSet
        System.out.println(h);

        // Checking if India is present or not
        System.out.println("\nHashSet contains India or not:"
                           + h.contains("India"));

        // Removing items from HashSet using remove()
        h.remove("Australia");

        // Printing the HashSet
        System.out.println("\nList after removing Australia:" + h);

        // Iterating over hash set items
        System.out.println("\nIterating over list:");
        Iterator<String> i = h.iterator();
        while (i.hasNext())
            System.out.println(i.next());
    }
}
```

Output:

```
[South Africa, Australia, India]

HashSet contains India or not:true

List after removing Australia:[South Africa, India]

Iterating over list:
South Africa
India
```

6. With the help of LinkedHashSet (Similar to LinkedHashMap, but maintains only keys, not pair)

```
 // Java program to demonstrate working of LinkedHashSet

import java.util.LinkedHashSet;
public class Demo
{
    public static void main(String[] args)
    {
```

```
        LinkedHashSet<String> linkedset =
                            new LinkedHashSet<String>();

        // Adding element to LinkedHashSet
        linkedset.add("A");
        linkedset.add("B");
        linkedset.add("C");
        linkedset.add("D");

        // This will not add new element as A already exists
        linkedset.add("A");
        linkedset.add("E");

        System.out.println("Size of LinkedHashSet = " +
                                    linkedset.size());
        System.out.println("Original LinkedHashSet:" + linkedset);
        System.out.println("Removing D from LinkedHashSet: " +
                            linkedset.remove("D"));
        System.out.println("Trying to Remove Z which is not "+
                            "present: " + linkedset.remove("Z"));
        System.out.println("Checking if A is present=" +
                            linkedset.contains("A"));
        System.out.println("Updated LinkedHashSet: " + linkedset);
    }
}
```

Output:

```
Size of LinkedHashSet = 5
Original LinkedHashSet:[A, B, C, D, E]
Removing D from LinkedHashSet: true
Trying to Remove Z which is not present: false
Checking if A is present=true
Updated LinkedHashSet: [A, B, C, E]
```

Source

https://www.geeksforgeeks.org/hashing-in-java/

Chapter 152

Hashing | Set 1 (Introduction)

Hashing | Set 1 (Introduction) - GeeksforGeeks

Suppose we want to design a system for storing employee records keyed using phone numbers. And we want following queries to be performed efficiently:

1. Insert a phone number and corresponding information.
2. Search a phone number and fetch the information.
3. Delete a phone number and related information.

We can think of using the following data structures to maintain information about different phone numbers.

1. Array of phone numbers and records.
2. Linked List of phone numbers and records.
3. Balanced binary search tree with phone numbers as keys.
4. Direct Access Table.

For **arrays and linked lists**, we need to search in a linear fashion, which can be costly in practice. If we use arrays and keep the data sorted, then a phone number can be searched in O(Logn) time using Binary Search, but insert and delete operations become costly as we have to maintain sorted order.

With **balanced binary search tree**, we get moderate search, insert and delete times. All of these operations can be guaranteed to be in O(Logn) time.

Another solution that one can think of is to use a **direct access table** where we make a big array and use phone numbers as index in the array. An entry in array is NIL if phone number is not present, else the array entry stores pointer to records corresponding to phone number. Time complexity wise this solution is the best among all, we can do all operations in O(1) time. For example to insert a phone number, we create a record with details of given phone number, use phone number as index and store the pointer to the created record in table.

This solution has many practical limitations. First problem with this solution is extra space required is huge. For example if phone number is n digits, we need O(m * 10^n) space for table where m is size of a pointer to record. Another problem is an integer in a programming language may not store n digits.

Due to above limitations Direct Access Table cannot always be used. **Hashing** is the solution that can be used in almost all such situations and performs extremely well compared to above data structures like Array, Linked List, Balanced BST in practice. With hashing we get O(1) search time on average (under reasonable assumptions) and O(n) in worst case.

Hashing is an improvement over Direct Access Table. The idea is to use hash function that converts a given phone number or any other key to a smaller number and uses the small number as index in a table called hash table.

Hash Function: A function that converts a given big phone number to a small practical integer value. The mapped integer value is used as an index in hash table. In simple terms, a hash function maps a big number or string to a small integer that can be used as index in hash table.
A good hash function should have following properties
1) Efficiently computable.
2) Should uniformly distribute the keys (Each table position equally likely for each key)

For example for phone numbers a bad hash function is to take first three digits. A better function is consider last three digits. Please note that this may not be the best hash function. There may be better ways.

Hash Table: An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.

Collision Handling: Since a hash function gets us a small number for a big key, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

- **Chaining:**The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.
- **Open Addressing:** In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table.

Next Posts:
Separate Chaining for Collision Handling
Open Addressing for Collision Handling

References:
MIT Video Lecture

IITD Video Lecture

"Introduction to Algorithms", Second Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein.

http://www.cs.princeton.edu/~rs/AlgsDS07/10Hashing.pdf

http://www.martinbroadhurst.com/articles/hash-table.html

Source

https://www.geeksforgeeks.org/hashing-set-1-introduction/

Chapter 153

Hashing | Set 2 (Separate Chaining)

Hashing | Set 2 (Separate Chaining) - GeeksforGeeks

We strongly recommend to refer below post as a prerequisite of this.
Hashing | Set 1 (Introduction)

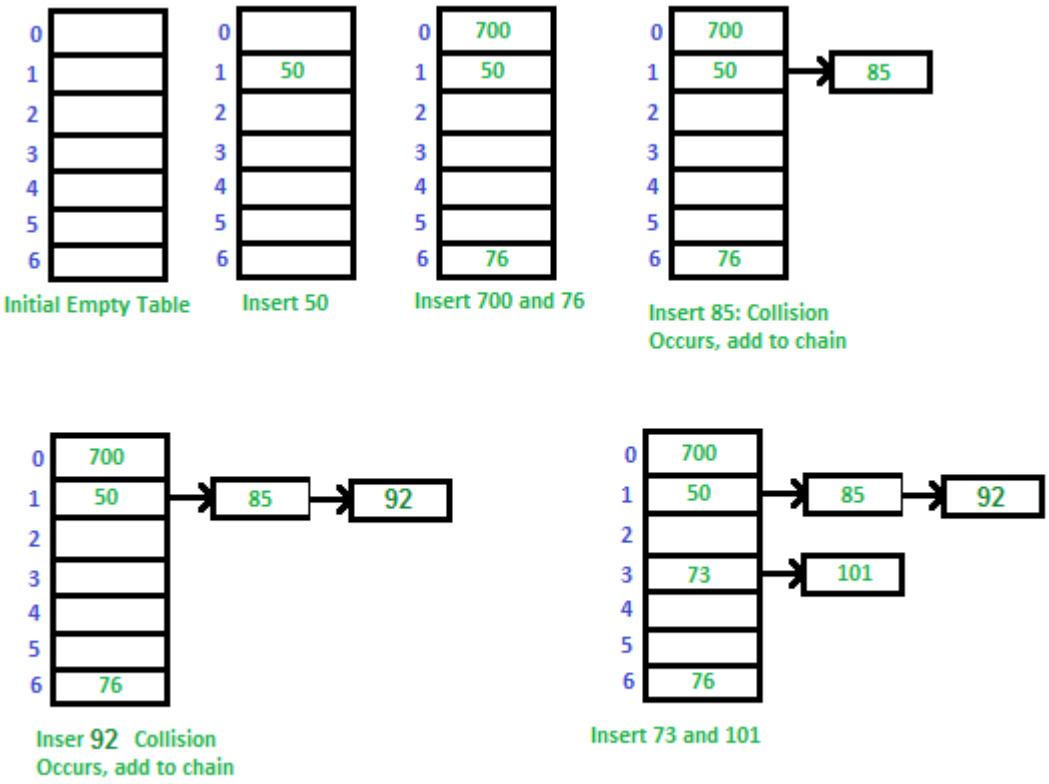
What is Collision?
Since a hash function gets us a small number for a key which is a big integer or string, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique.

What are the chances of collisions with large table?
Collisions are very likely even if we have big table to store keys. An important observation is Birthday Paradox. With only 23 persons, the probability that two people have same birthday is 50%.

How to handle Collisions?
There are mainly two methods to handle collision:
1) Separate Chaining
2) Open Addressing
In this article, only separate chaining is discussed. We will be discussing Open addressing in next post.

Separate Chaining:
The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Let us consider a simple hash function as "**key mod 7**" and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



C++ program for hashing with chaining

Advantages:
1) Simple to implement.
2) Hash table never fills up, we can always add more elements to chain.
3) Less sensitive to the hash function or load factors.
4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Disadvantages:
1) Cache performance of chaining is not good as keys are stored using linked list. Open addressing provides better cache performance as everything is stored in same table.
2) Wastage of Space (Some Parts of hash table are never used)
3) If the chain becomes long, then search time can become O(n) in worst case.
4) Uses extra space for links.

Performance of Chaining:
Performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of table (simple uniform hashing).

```
m = Number of slots in hash table
```

```
n = Number of keys to be inserted in hash table

Load factor   = n/m

Expected time to search = O(1 +  )

Expected time to insert/delete = O(1 +  )

Time complexity of search insert and delete is
O(1) if    is O(1)
```

Next Post:
Open Addressing for Collision Handling

References:
http://courses.csail.mit.edu/6.006/fall09/lecture_notes/lecture05.pdf

Source

https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/

Chapter 154

Hashing | Set 3 (Open Addressing)

Hashing | Set 3 (Open Addressing) - GeeksforGeeks

We strongly recommend to refer below post as a prerequisite of this.
Hashing | Set 1 (Introduction)
Hashing | Set 2 (Separate Chaining)

Open Addressing
Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

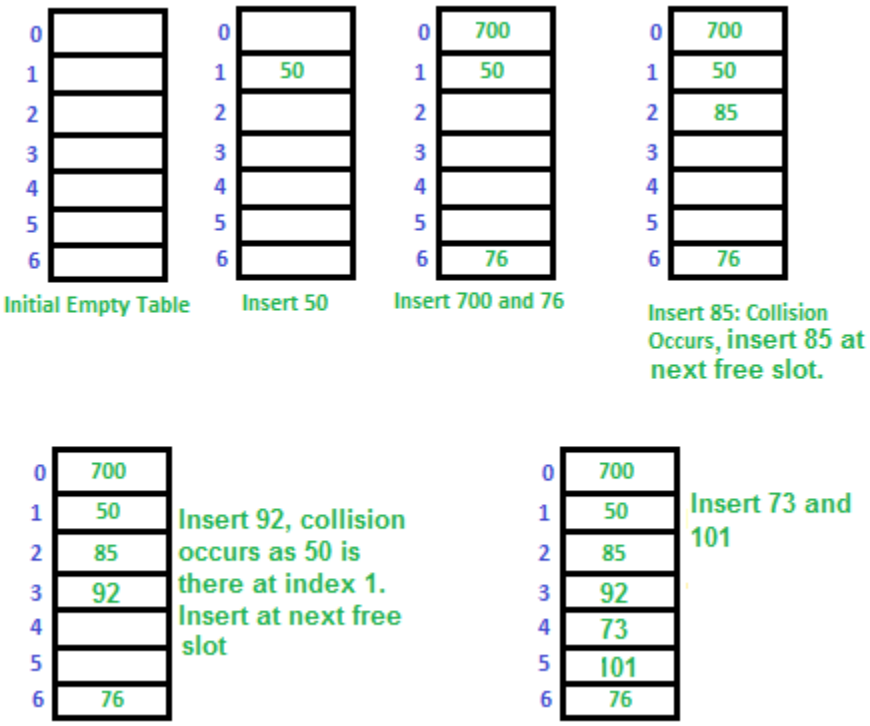
Delete(k): ***Delete operation is interesting***. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".
Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done following ways:

a) Linear Probing: In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also.
let **hash(x)** be the slot index computed using hash function and **S** be the table size

```
If slot hash(x) % S is full, then we try (hash(x) + 1) % S
If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S
If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S
.............................................
.............................................
```

Let us consider a simple hash function as "key mod 7" and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



Clustering: The main problem with linear probing is clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

b) Quadratic Probing We look for i^2'th slot in i'th iteration.

```
let hash(x) be the slot index computed using hash function.
If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S
If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S
If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S
.................................................
.................................................
```

c)Double Hashing We use another hash function hash2(x) and look for i*hash2(x) slot in i'th rotation.

```
let hash(x) be the slot index computed using hash function.
If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S
If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S
If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S
.................................................
.................................................
```

See thisfor step by step diagrams.

Comparison of above three:
Linear probing has the best cache performance but suffers from clustering. One more advantage of Linear probing is easy to compute.

Quadratic probing lies between the two in terms of cache performance and clustering.

Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.

S.No.

Seperate Chaining

Open Addressing

1.

Chaining is Simpler to implement.

Open Addressing requires more computation.

2.

In chaining, Hash table never fills up, we can always add more elements to chain.

In open addressing, table may become full.

3.

Chaining is Less sensitive to the hash function or load factors.

Open addressing requires extra care for to avoid clustering and load factor.

4.

Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Open addressing is used when the frequency and number of keys is known.

5.

Cache performance of chaining is not good as keys are stored using linked list.

Open addressing provides better cache performance as everything is stored in the same table.

6.

Wastage of Space (Some Parts of hash table in chaining are never used).

In Open addressing, a slot can be used even if an input doesn't map to it.

7.

Chaining uses extra space for links.

No links in Open addressing

Performance of Open Addressing:
Like Chaining, the performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of the table (simple uniform hashing)

```
m = Number of slots in the hash table
n = Number of keys to be inserted in the hash table

Load factor   = n/m  ( < 1 )

Expected time to search/insert/delete < 1/(1 -  )

So Search, Insert and Delete take (1/(1 -  )) time
```

References:
http://courses.csail.mit.edu/6.006/fall11/lectures/lecture10.pdf
https://www.cse.cuhk.edu.hk/irwin.king/_media/teaching/csc2100b/tu6.pdf

Improved By : PulkitGoel2

Source

https://www.geeksforgeeks.org/hashing-set-3-open-addressing/

Chapter 155

Hashtables Chaining with Doubly Linked Lists

Hashtables Chaining with Doubly Linked Lists - GeeksforGeeks

Prerequisite – Hashing Introduction, Hashtable using Singly Linked List & Implementing our Own Hash Table with Separate Chaining in Java

Implementing hash table using Chaining through Doubly Linked List is similar to implementingHashtable using Singly Linked List. The only difference is that every node of Linked List has the address of both, the next and the previous node. This will speed up the process of adding and removing elements from the list, hence the time complexity will be reduced drastically.
Example:

> If we have a Singly linked list:
>
> ```
> 1->2->3->4
> ```
>
> If we are at 3 and there is a need to remove it, then 2 need to be linked with 4 and as from 3, 2 can't be accessed as it is singly linked list. So, the list has to be traversed again i.e O(n), but if we have doubly linked list i.e.
>
> ```
> 1234
> ```
>
> 2 & 4 can be accessed from 3, hence in O(1), 3 can be removed.

Below is the implementation of the above approach:

```
 // C++ implementation of Hashtable
// using doubly linked list
#include <bits/stdc++.h>
using namespace std;

const int tablesize = 25;

// declaration of node
struct hash_node {
    int val, key;
    hash_node* next;
    hash_node* prev;
};

// hashmap's declaration
class HashMap {
public:
    hash_node **hashtable, **top;

    // constructor
    HashMap()
    {
        // create a empty hashtable
        hashtable = new hash_node*[tablesize];
        top = new hash_node*[tablesize];
        for (int i = 0; i < tablesize; i++) {
            hashtable[i] = NULL;
            top[i] = NULL;
        }
    }

    // destructor
    ~HashMap()
    {
        delete[] hashtable;
    }

    // hash function definition
    int HashFunc(int key)
    {
        return key % tablesize;
    }

    // searching method
    void find(int key)
    {
        // Applying hashFunc to find
        // index for given key
```

```
    int hash_val = HashFunc(key);
    bool flag = false;
    hash_node* entry = hashtable[hash_val];

    // if hashtable at that index has some
    // values stored
    if (entry != NULL) {
        while (entry != NULL) {
            if (entry->key == key) {
                flag = true;
            }
            if (flag) {
                cout << "Element found at key "
                     << key << ": ";
                cout << entry->val << endl;
            }
            entry = entry->next;
        }
    }
    if (!flag)
        cout << "No Element found at key "
             << key << endl;
}

// removing an element
void remove(int key)
{
    // Applying hashFunc to find
    // index for given key
    int hash_val = HashFunc(key);
    hash_node* entry = hashtable[hash_val];
    if (entry->key != key || entry == NULL) {
        cout << "Couldn't find any element at this key "
             << key << endl;
        return;
    }

    // if some values are present at that key &
    // traversing the list and removing all values
    while (entry != NULL) {
        if (entry->next == NULL) {
            if (entry->prev == NULL) {
                hashtable[hash_val] = NULL;
                top[hash_val] = NULL;
                delete entry;
                break;
            }
            else {
```

```
                top[hash_val] = entry->prev;
                top[hash_val]->next = NULL;
                delete entry;
                entry = top[hash_val];
            }
        }
        entry = entry->next;
    }
    cout << "Element was successfully removed at the key "
         << key << endl;
}

// inserting method
void add(int key, int value)
{
    // Applying hashFunc to find
    // index for given key
    int hash_val = HashFunc(key);
    hash_node* entry = hashtable[hash_val];

    // if key has no value stored
    if (entry == NULL) {
        // creating new node
        entry = new hash_node;
        entry->val = value;
        entry->key = key;
        entry->next = NULL;
        entry->prev = NULL;
        hashtable[hash_val] = entry;
        top[hash_val] = entry;
    }

    // if some values are present
    else {
        // traversing till the end of
        // the list
        while (entry != NULL)
            entry = entry->next;

        // creating the new node
        entry = new hash_node;
        entry->val = value;
        entry->key = key;
        entry->next = NULL;
        entry->prev = top[hash_val];
        top[hash_val]->next = entry;
        top[hash_val] = entry;
    }
```

```
        cout << "Value " << value << " was successfully"
                " added at key " << key << endl;
    }
};

// Driver Code
int main()
{
    HashMap hash;
    hash.add(4, 5);
    hash.find(4);
    hash.remove(4);
    return 0;
}
```

Output:

```
Value 5 was successfully added at key 4
Element found at key 4: 5
Element was successfully removed at the key 4
```

Source

https://www.geeksforgeeks.org/hashtables-chaining-with-doubly-linked-lists/

Chapter 156

How to check if two given sets are disjoint?

How to check if two given sets are disjoint? - GeeksforGeeks

Given two sets represented by two arrays, how to check if the given two sets are disjoint or not? It may be assumed that the given arrays have no duplicates.

Difficulty Level: Rookie

```
Input: set1[] = {12, 34, 11, 9, 3}
       set2[] = {2, 1, 3, 5}
Output: Not Disjoint
3 is common in two sets.

Input: set1[] = {12, 34, 11, 9, 3}
       set2[] = {7, 2, 1, 5}
Output: Yes, Disjoint
There is no common element in two sets.
```

We strongly recommend to minimize your browser and try this yourself first.
There are plenty of methods to solve this problem, it's a good test to check how many solutions you can guess.

Method 1 (Simple)
Iterate through every element of first set and search it in other set, if any element is found, return false. If no element is found, return tree. Time complexity of this method is O(mn).

Following is implementation of above idea.

C++

```
// A Simple C++ program to check if two sets are disjoint
```

```
#include<iostream>
using namespace std;

// Returns true if set1[] and set2[] are disjoint, else false
bool areDisjoint(int set1[], int set2[], int m, int n)
{
    // Take every element of set1[] and search it in set2
    for (int i=0; i<m; i++)
      for (int j=0; j<n; j++)
         if (set1[i] == set2[j])
            return false;

    // If no element of set1 is present in set2
    return true;
}

// Driver program to test above function
int main()
{
    int set1[] = {12, 34, 11, 9, 3};
    int set2[] = {7, 2, 1, 5};
    int m = sizeof(set1)/sizeof(set1[0]);
    int n = sizeof(set2)/sizeof(set2[0]);
    areDisjoint(set1, set2, m, n)? cout << "Yes" : cout << " No";
    return 0;
}
```

Java

```
 // Java program to check if two sets are disjoint

public class disjoint1
{
    // Returns true if set1[] and set2[] are
    // disjoint, else false
    boolean aredisjoint(int set1[], int set2[])
    {
         // Take every element of set1[] and
         // search it in set2
        for (int i = 0; i < set1.length; i++)
        {
            for (int j = 0; j < set2.length; j++)
            {
                if (set1[i] == set2[j])
                    return false;
            }
        }
        // If no element of set1 is present in set2
```

```
        return true;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        disjoint1 dis = new disjoint1();
        int set1[] = { 12, 34, 11, 9, 3 };
        int set2[] = { 7, 2, 1, 5 };

        boolean result = dis.aredisjoint(set1, set2);
        if (result)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by Rishabh Mahrsee
```

Python

```
 # A Simple python 3 program to check
# if two sets are disjoint

# Returns true if set1[] and set2[] are disjoint, else false
def areDisjoint(set1, set2, m, n):
    # Take every element of set1[] and search it in set2
    for i in range(0, m):
        for j in range(0, n):
            if (set1[i] == set2[j]):
                return False

    # If no element of set1 is present in set2
    return True


# Driver program
set1 = [12, 34, 11, 9, 3]
set2 = [7, 2, 1, 5]
m = len(set1)
n = len(set2)
print("yes") if areDisjoint(set1, set2, m, n) else(" No")

# This code ia contributed by Smitha Dinesh Semwal
```

Output :

```
Yes
```

Method 2 (Use Sorting and Merging)
1) Sort first and second sets.
2) Use merge like process to compare elements.

Following is implementation of above idea.

C++

```
 // A Simple C++ program to check if two sets are disjoint
#include<iostream>
#include<algorithm>
using namespace std;

// Returns true if set1[] and set2[] are disjoint, else false
bool areDisjoint(int set1[], int set2[], int m, int n)
{
    // Sort the given two sets
    sort(set1, set1+m);
    sort(set2, set2+n);

    // Check for same elements using merge like process
    int i = 0, j = 0;
    while (i < m && j < n)
    {
        if (set1[i] < set2[j])
            i++;
        else if (set2[j] < set1[i])
            j++;
        else /* if set1[i] == set2[j] */
            return false;
    }

    return true;
}

// Driver program to test above function
int main()
{
    int set1[] = {12, 34, 11, 9, 3};
    int set2[] = {7, 2, 1, 5};
    int m = sizeof(set1)/sizeof(set1[0]);
    int n = sizeof(set2)/sizeof(set2[0]);
    areDisjoint(set1, set2, m, n)? cout << "Yes" : cout << " No";
    return 0;
}
```

Java

```
 // Java program to check if two sets are disjoint

import java.util.Arrays;

public class disjoint1
{
    // Returns true if set1[] and set2[] are
    // disjoint, else false
    boolean aredisjoint(int set1[], int set2[])
    {
        int i=0,j=0;

        // Sort the given two sets
        Arrays.sort(set1);
        Arrays.sort(set2);

        // Check for same elements using
        // merge like process
        while(i<set1.length && j<set2.length)
        {
            if(set1[i]<set2[j])
                i++;
            else if(set1[i]>set2[j])
                j++;
            else
                return false;

        }
        return true;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        disjoint1 dis = new disjoint1();
        int set1[] = { 12, 34, 11, 9, 3 };
        int set2[] = { 7, 2, 1, 5 };

        boolean result = dis.aredisjoint(set1, set2);
        if (result)
            System.out.println("YES");
        else
            System.out.println("NO");
    }
}
```

```
// This code is contributed by Rishabh Mahrsee
```

Python

```
 # A Simple Python 3 program to check
# if two sets are disjoint

# Returns true if set1[] and set2[]
# are disjoint, else false
def areDisjoint(set1, set2, m, n):
    # Sort the given two sets
    set1.sort()
    set2.sort()

    # Check for same elements
    # using merge like process
    i = 0; j = 0
    while (i < m and j < n):

        if (set1[i] < set2[j]):
            i += 1
        elif (set2[j] < set1[i]):
            j += 1
        else: # if set1[i] == set2[j]
            return False
    return True


# Driver Code
set1 = [12, 34, 11, 9, 3]
set2 = [7, 2, 1, 5]
m = len(set1)
n = len(set2)

print("Yes") if areDisjoint(set1, set2, m, n) else print("No")

# This code is contributed by Smitha Dinesh Semwal
```

Output :

```
Yes
```

Time complexity of above solution is O(mLogm + nLogn).

The above solution first sorts both sets, then takes O(m+n) time to find intersection. If we are given that the input sets are sorted, then this method is best among all.

Method 3 (Use Sorting and Binary Search)
This is similar to method 1. Instead of linear search, we use Binary Search.

1) Sort first set.
2) Iterate through every element of second set, and use binary search to search every element in first set. If element is found return it.

Time complexity of this method is O(mLogm + nLogm)

Method 4 (Use Binary Search Tree)
1) Create a self balancing binary search tree (Red Black, AVL, Splay, etc) of all elements in first set.
2) Iterate through all elements of second set and search every element in the above constructed Binary Search Tree. If element is found, return false.
3) If all elements are absent, return true.

Time complexity of this method is O(mLogm + nLogm).

Method 5 (Use Hashing)
1) Create an empty hash table.
2) Iterate through the first set and store every element in hash table.
3) Iterate through second set and check if any element is present in hash table. If present, then return false, else ignore the element.
4) If all elements of second set are not present in hash table, return true.

Following are C++ and Java implementation of this method.

C/C++

```
 #include<bits/stdc++.h>
using namespace std;

/* C++ program to check if two sets are distinct or not */
// This function prints all distinct elements
bool areDisjoint(int set1[], int set2[], int n1, int n2)
{
    // Creates an empty hashset
    set<int> myset;

    // Traverse the first set and store its elements in hash
    for (int i = 0; i < n1; i++)
        myset.insert(set1[i]);

    // Traverse the second set and check if any element of it
    // is already in hash or not.
    for (int i = 0; i < n2; i++)
        if (myset.find(set2[i]) != myset.end())
            return false;

    return true;
}

// Driver method to test above method
int main()
```

```
{
    int set1[] = {10, 5, 3, 4, 6};
    int set2[] = {8, 7, 9, 3};

    int n1 = sizeof(set1) / sizeof(set1[0]);
    int n2 = sizeof(set2) / sizeof(set2[0]);
    if (areDisjoint(set1, set2, n1, n2))
        cout << "Yes";
    else
        cout << "No";
}
//This article is contributed by Chhavi
```

JAVA

```
 /* Java program to check if two sets are distinct or not */
import java.util.*;

class Main
{
    // This function prints all distinct elements
    static boolean areDisjoint(int set1[], int set2[])
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the first set and store its elements in hash
        for (int i=0; i<set1.length; i++)
            set.add(set1[i]);

        // Traverse the second set and check if any element of it
        // is already in hash or not.
        for (int i=0; i<set2.length; i++)
            if (set.contains(set2[i]))
                return false;

        return true;
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int set1[] = {10, 5, 3, 4, 6};
        int set2[] = {8, 7, 9, 3};
        if (areDisjoint(set1, set2))
            System.out.println("Yes");
        else
            System.out.println("No");
```

```
    }
}
```

Output:

```
No
```

Time complexity of the above implementation is O(m+n) under the assumption that hash set operations like add() and contains() work in O(1) time.

This article is contributed by **Rajeev**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/check-two-given-sets-disjoint/

Chapter 157

How to store a password in database?

How to store a password in database? - GeeksforGeeks

Most of the web applications require their users to authenticate themselves by asking them username and password. They compare the user supplied credentials with the data stored in their database and if the credentials match, the user is granted access. Sounds good! But what will happen if the database in which the website is storing your passwords gets compromised?
This article covers various techniques of storing passwords in the database.

According to naked security, 55% of the net users use the same password for most of the websites! It implies that if the website storing your password in plain text gets compromised, hacker is not only able to gain access of your account on that website but all your social media, email, forums etc accounts in which you are using the same password!

Well, many must be wondering that if the database is exposed to the hacker then what can be done? The hacker has access to all the information. WRONG!! There are many ways through which the process of retrieving password from the database can be made cumbersome for the hacker. Even then the developers tend to ignore the basic guidelines and store the passwords in plain text.There are over 30% websites which store your passwords in plain text (including some reputed sites also). If the website is storing your password in plain text then no matter how strong password you choose, you are not safe!

Storing plain text passwords in the database is a sin.

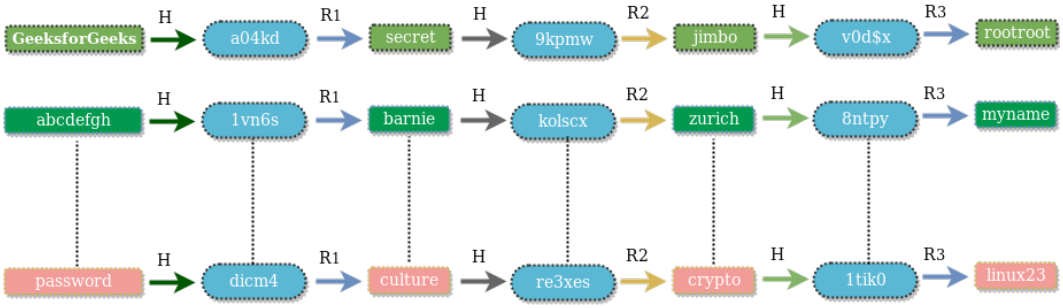
One might also think that if not plain text then we must encrypt the password and then store. It is also a terrible idea. Encryption functions provide one-one mapping between input and output and they are always reversible. If the hacker gets the key, he will

be able to decrypt the passwords. The better way would be to use a one way cryptographic hash function. Hash function provides a many-one mapping between input and output and it is practically impossible to reverse a output. A good cryptographic hash function has lesser number of Collisions (i.e for different input values to the function it is difficult to get the same output). Collisions cannot be completely avoided because of pigeonhole principle. For hashing passwords we can assume that the hash function will generate unique output i.e for no two different passwords we will get a same hash value.

Some of the popular cryptographic hash functions are MD5 and SHA1. Instead of storing plain text password in the database one way is to store the hash of the password. You might be thinking that if we cannot get the actual password back from the hash then how are we going to verify the credentials that the user entered? It's simple, apply the same hash function on the password which user entered and then compare it with the hash stored in the database. If both hashes match then the user is authenticated (since hash of same input will give same output). Now if the attacker is able to get database access, he will be only able to view the hashed output and not the actual password.

Using cryptographic hash function is better than storing plain text password.

Hackers are smart guys and once they came to know that developers are storing hashed passwords, they pre-computed hash of large number of words (from a popular word list or dictionary words). They created a table of words and their corresponding hashes. This table is known as Rainbow Table and it is readily available online. They can use this table to reverse lookup the actual password by comparing the hashes obtained from the database. Hence it is very important to have a **strong password** since the possibility of your password appearing in the word list becomes less.



Simply storing the hash of a password is not going to help anymore. Processing power has increased drastically with the introduction of GPUs and CUDA, OpenCL libraries. A fast GPU can generate millions of MD5/SHA1 hashes in one second. Hence a hacker can easily generate large number of hashes by brute-forcing various possible combinations and can compare it with the hashes stored in the database to extract the actual password.

Even hashed passwords are not secure! Surprised?

Don't loose hope! There is still something that developers can do to keep your pass-

words away from prying eyes of the hackers. Make the passwords delicious by adding some salt to them! Yeah, right..! Add a salt. A salt is random data that is concatenated with your password before sending it as the input of the hashing function.

For example :
If your password is *abc* and the salt is *!ZaP0#8*, the result of ***hashFunction('abc!ZaP0#8')*** will be stored in the database instead of ***hashFunction('abc')***.
Hence the rainbow table attacks won't be effective now as the probability that rainbow table contains hash of *'abc!ZaP0#8'* is meager (because generally rainbow tables are constructed from common words, dictionary words etc). Salt is not stored in the database and only present in the application configuration file which is not accessible to outer world. Gaining access to the source files is difficult than gaining access to the database.

The above salting method is static. We have one fixed salt for all the passwords. To authenticate the user, first concatenate the fixed salt to the user supplied input (password) and then pass the value to the hashing function and compare it with the value stored in the database. However this approach is still vulnerable to brute-force and if the attacker is able to get the static salt he can use the old attack methodology by concatenating the salt in every word.

A better approach would be to use a dynamic salt. For each user a new salt is generated by cryptographically strong random string generator. The password entered by user is concatenated with a random generated salt as well as a static salt. The concatenated string is passed as the input of hashing function. The result obtained is stored in database. Dynamic salt is required to be stored in the database since it is different for different users. When the user is to be authenticated, first the value of dynamic salt for that user is fetched from the database, it is concatenated with user supplied input and the static salt. The result is compared with the hash stored in the database.

If the database is compromised the hacker will not only get your password hashes but also the dynamic salt used. You might be wondering then what is the advantage of dynamic salt over static salt if attacker has dynamic salt? Even if the attacker has dynamic salt he needs to create a new hash-table (or rainbow table) for each and every user present in the database (as per dynamic salt). This is a lot more expensive operation than creating just one table for all the users.

The above approach is quite good to slow down a hacker. However it is recommended to use algorithms like *bcrypt* and *scrypt* instead of MD5/SHA1. Bcrypt is a hashing algorithm based on Blowfish. It requires you to specify a cost/work factor. The work factor makes the overall process slower and hence time taken to generate hash-table would increase multiple times.

References :
https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/

Source

https://www.geeksforgeeks.org/store-password-database/

Chapter 158

Implementing our Own Hash Table with Separate Chaining in Java

Implementing our Own Hash Table with Separate Chaining in Java - GeeksforGeeks

Every data structure has its own special characteristics for example a BST is used when quick searching of an element (in log(n)) is required. A heap or a priority queue is used when the minimum or maximum element needs to be fetched in constant time. Similarly a hash table is used to fetch, add and remove an element in constant time. It is necessary for anyone to be clear with the working of a hash table before moving on to the implementation aspect. So here is a brief background on the working of hash table and also it should be noted that we will be using Hash Map and Hash Table terminology interchangeably though in Java HashTables are thread safe while HashMaps are not.

The code we are going to implement is available at Link 1 and Link2

But it is strongly recommended that one must read this blog completely and try and decipher the nitty gritty of what goes into implementing a hash map and then try to write the code yourself.

Background

Every hash-table stores data in the form of (key, value) combination. Interestingly every key is unique in a Hash Table but values can repeat which means values can be same for different keys present in it. Now as we observe in an array to fetch a value we provide the position/index corresponding to the value in that array. In a Hash Table, instead of an index we use a key to fetch the value corresponding to that key. Now the entire process is described below

Every time a key is generated. The key is passed to a hash function. Every hash function has two parts a **Hash code and a Compressor**.

Hash code is an Integer number (random or nonrandom). In Java every Object has its own hash code. We will use the hash code generated by JVM in our hash function and to

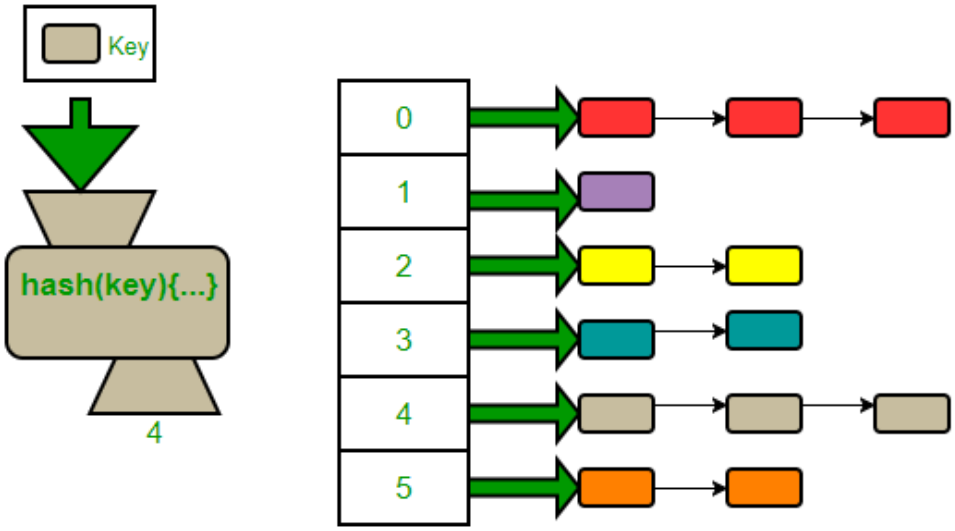
compress the hash code we modulo(%) the hash code by size of the hash table. *So modulo operator is compressor in our implementation.*

The entire process ensures that for any key, we get an integer position within the size of the Hash Table to insert the corresponding value.

So the process is simple, user gives a (key, value) pair set as input and based on the value generated by hash function an index is generated to where the value corresponding to the particular key is stored. So whenever we need to fetch a value corresponding to a key that is just O(1).

This picture stops being so rosy and perfect when the concept of hash collision is introduced. Imagine for different key values same block of hash table is allocated now where do the previously stored values corresponding to some other previous key go. We certainly can't replace it .That will be disastrous! To resolve this issue we will use Separate Chaining Technique, Please note there are other open addressing techniques like double hashing and linear probing whose efficiency is almost same as to that of separate chaining and you can read more about them at Link 1 Link 2 Link3

Now what we do is make a linked list corresponding to the particular bucket of the Hash Table, to accommodate all the values corresponding to different keys who map to the same bucket.



Now there may be a scenario that all the keys get mapped to the same bucket and we have a linked list of n(size of hash table) size from one single bucket, with all the other buckets empty and this is the worst case where a hash table acts a linked list and searching is O(n).So what do we do ?

Load Factor

If n be the total number of buckets we decided to fill initially say 10 and let's say 7 of them got filled now, so the load factor is 7/10=0.7.

In our implementation whenever we add a key value pair to the Hash Table we check the load factor if it is greater than 0.7 we double the size of our hash table.

Implementation

Hash Node Data Type

We will try to make a generic map without putting any restrictions on the data type of the key and the value . Also every hash node needs to know the next node it is pointing to in the linked list so a next pointer is also required.

The functions we plan to keep in our hash map are

- **get(K key)** : returns the value corresponding to the key if the key is present in **HT** (**H**ast **T**able)
- **getSize()** : return the size of the HT
- **add()** : adds new valid key, value pair to the HT, if already present updates the value
- **remove()** : removes the key, value pair
- **isEmpty()** : returns true if size is zero

Every Hash Map must have an array list/linked list with an initial size and a bucket size which gets increased by unity every time a key, value pair is added and decreased by unity every time a node is deleted

```
ArrayList<HashNode<K, V>> bucket = new ArrayList<>();
```

A **Helper Function** is implemented to get the index of the key, to avoid redundancy in other functions like get, add and remove. This function uses the in built java function to generate a hash code and we compress the hash code by the size of the HT so that the index is within the range of the size of the HT

get()
The get function just takes a key as an input and returns the corresponding value if the key is present in the table otherwise returns null. Steps are:

- Retrieve the input key to find the index in the HT
- Traverse the liked list corresponding to the HT, if you find the value then return it else if you fully traverse the list without returning it means the value is not present in the table and can't be fetched so return null

remove()

- Fetch the index corresponding to the input key using the helper function
- The traversal of linked list similar like in get() but what is special here is that one needs to remove the key along with finding it and two cases arise
- If the key to be removed is present at the head of the linked list
- If the key to be removed is not present at head but somewhere else

add()
Now to the most interesting and challenging function of this entire implementation.It is interesting because we need to dynamically increase the size of our list when load factor is above the value we specified.

- Just like remove steps till traversal and adding and two cases (addition at head spot or non-head spot) remain the same.
- Towards the end if load factor is greater than 0.7
- We double the size of the array list and then recursively call add function on existing keys because in our case hash value generated uses the size of the array to compress the inbuilt JVM hash code we use ,so we need to fetch new indices for the existing keys. This is very important to understand please re read this paragraph till you get a hang of what is happening in the add function.

Java does in its own implementation of Hash Table uses Binary Search Tree if linked list corresponding to a particular bucket tend to get too long.

```
 // Java program to demonstrate implementation of our
// own hash table with chaining for collision detection
import java.util.ArrayList;

// A node of chains
class HashNode<K, V>
{
    K key;
    V value;

    // Reference to next node
    HashNode<K, V> next;

    // Constructor
    public HashNode(K key, V value)
    {
        this.key = key;
        this.value = value;
    }
}

// Class to represent entire hash table
class Map<K, V>
{
    // bucketArray is used to store array of chains
    private ArrayList<HashNode<K, V>> bucketArray;

    // Current capacity of array list
    private int numBuckets;
```

```
// Current size of array list
private int size;

// Constructor (Initializes capacity, size and
// empty chains.
public Map()
{
    bucketArray = new ArrayList<>();
    numBuckets = 10;
    size = 0;

    // Create empty chains
    for (int i = 0; i < numBuckets; i++)
        bucketArray.add(null);
}

public int size() { return size; }
public boolean isEmpty() { return size() == 0; }

// This implements hash function to find index
// for a key
private int getBucketIndex(K key)
{
    int hashCode = key.hashCode();
    int index = hashCode % numBuckets;
    return index;
}

// Method to remove a given key
public V remove(K key)
{
    // Apply hash function to find index for given key
    int bucketIndex = getBucketIndex(key);

    // Get head of chain
    HashNode<K, V> head = bucketArray.get(bucketIndex);

    // Search for key in its chain
    HashNode<K, V> prev = null;
    while (head != null)
    {
        // If Key found
        if (head.key.equals(key))
            break;

        // Else keep moving in chain
        prev = head;
        head = head.next;
```

```
    }

    // If key was not there
    if (head == null)
        return null;

    // Reduce size
    size--;

    // Remove key
    if (prev != null)
        prev.next = head.next;
    else
        bucketArray.set(bucketIndex, head.next);

    return head.value;
}

// Returns value for a key
public V get(K key)
{
    // Find head of chain for given key
    int bucketIndex = getBucketIndex(key);
    HashNode<K, V> head = bucketArray.get(bucketIndex);

    // Search key in chain
    while (head != null)
    {
        if (head.key.equals(key))
            return head.value;
        head = head.next;
    }

    // If key not found
    return null;
}

// Adds a key value pair to hash
public void add(K key, V value)
{
    // Find head of chain for given key
    int bucketIndex = getBucketIndex(key);
    HashNode<K, V> head = bucketArray.get(bucketIndex);

    // Check if key is already present
    while (head != null)
    {
        if (head.key.equals(key))
```

```
            {
                head.value = value;
                return;
            }
            head = head.next;
        }

        // Insert key in chain
        size++;
        head = bucketArray.get(bucketIndex);
        HashNode<K, V> newNode = new HashNode<K, V>(key, value);
        newNode.next = head;
        bucketArray.set(bucketIndex, newNode);

        // If load factor goes beyond threshold, then
        // double hash table size
        if ((1.0*size)/numBuckets >= 0.7)
        {
            ArrayList<HashNode<K, V>> temp = bucketArray;
            bucketArray = new ArrayList<>();
            numBuckets = 2 * numBuckets;
            size = 0;
            for (int i = 0; i < numBuckets; i++)
                bucketArray.add(null);

            for (HashNode<K, V> headNode : temp)
            {
                while (headNode != null)
                {
                    add(headNode.key, headNode.value);
                    headNode = headNode.next;
                }
            }
        }
    }

    // Driver method to test Map class
    public static void main(String[] args)
    {
        Map<String, Integer>map = new Map<>();
        map.add("this",1 );
        map.add("coder",2 );
        map.add("this",4 );
        map.add("hi",5 );
        System.out.println(map.size());
        System.out.println(map.remove("this"));
        System.out.println(map.remove("this"));
        System.out.println(map.size());
```

```
        System.out.println(map.isEmpty());
    }
}
```

Output :

```
3
4
null
2
false
```

The entire code is available at https://github.com/ishaan007/Data-Structures/blob/master/HashMaps/Map.java

Source

https://www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/

Chapter 159

Implementing own Hash Table with Open Addressing Linear Probing in C++

Implementing own Hash Table with Open Addressing Linear Probing in C++ - GeeksforGeeks

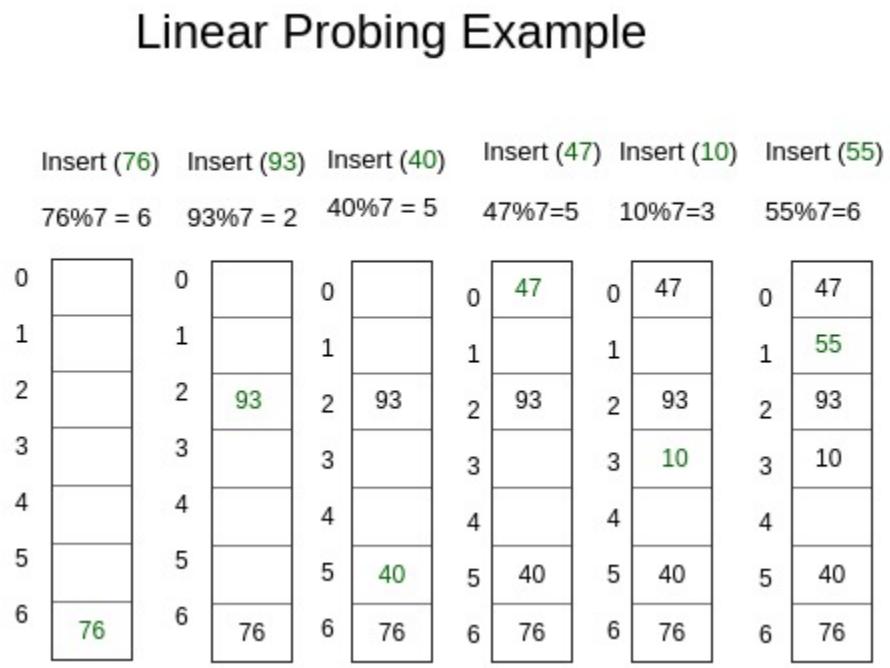
Prerequisite – Hashing Introduction, Implementing our Own Hash Table with Separate Chaining in Java

In Open Addressing, all elements are stored in the hash table itself. So at any point, size of table must be greater than or equal to total number of keys (Note that we can increase table size by copying old data if needed).

- **Insert(k)** – Keep probing until an empty slot is found. Once an empty slot is found, insert k.
- **Search(k)** – Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
- **Delete(k)** – Delete operation is interesting. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".

Here, to mark a node deleted we have used **dummy node** with key and value -1.
Insert can insert an item in a deleted slot, but search doesn't stop at a deleted slot.

The entire process ensures that for any key, we get an integer position within the size of the Hash Table to insert the corresponding value.
So the process is simple, user gives a (key, value) pair set as input and based on the value generated by hash function an index is generated to where the value corresponding to the particular key is stored. So whenever we need to fetch a value corresponding to a key that is just O(1).



Code –

```
 #include<bits/stdc++.h>
using namespace std;

//template for generic type
template<typename K, typename V>

//Hashnode class
class HashNode
{
    public:
    V value;
    K key;

    //Constructor of hashnode
    HashNode(K key, V value)
    {
        this->value = value;
        this->key = key;
    }
};

//template for generic type
template<typename K, typename V>
```

```
//Our own Hashmap class
class HashMap
{
    //hash element array
    HashNode<K,V> **arr;
    int capacity;
    //current size
    int size;
    //dummy node
    HashNode<K,V> *dummy;

    public:
    HashMap()
    {
        //Initial capacity of hash array
        capacity = 20;
        size=0;
        arr = new HashNode<K,V>*[capacity];

        //Initialise all elements of array as NULL
        for(int i=0 ; i < capacity ; i++)
            arr[i] = NULL;

        //dummy node with value and key -1
        dummy = new HashNode<K,V>(-1, -1);
    }
    // This implements hash function to find index
    // for a key
    int hashCode(K key)
    {
        return key % capacity;
    }

    //Function to add key value pair
    void insertNode(K key, V value)
    {
        HashNode<K,V> *temp = new HashNode<K,V>(key, value);

        // Apply hash function to find index for given key
        int hashIndex = hashCode(key);

        //find next free space
        while(arr[hashIndex] != NULL && arr[hashIndex]->key != key
               && arr[hashIndex]->key != -1)
        {
            hashIndex++;
            hashIndex %= capacity;
        }
```

```
    //if new node to be inserted increase the current size
    if(arr[hashIndex] == NULL || arr[hashIndex]->key == -1)
        size++;
    arr[hashIndex] = temp;
}

//Function to delete a key value pair
V deleteNode(int key)
{
    // Apply hash function to find index for given key
    int hashIndex = hashCode(key);

    //finding the node with given key
    while(arr[hashIndex] != NULL)
    {
        //if node found
        if(arr[hashIndex]->key == key)
        {
            HashNode<K,V> *temp = arr[hashIndex];

            //Insert dummy node here for further use
            arr[hashIndex] = dummy;

            // Reduce size
            size--;
            return temp->value;
        }
        hashIndex++;
        hashIndex %= capacity;

    }

    //If not found return null
    return NULL;
}

//Function to search the value for a given key
V get(int key)
{
    // Apply hash function to find index for given key
    int hashIndex = hashCode(key);

    //finding the node with given key
    while(arr[hashIndex] != NULL)
    {
        //if node found return its value
        if(arr[hashIndex]->key == key)
```

```
                return arr[hashIndex]->value;
            hashIndex++;
            hashIndex %= capacity;
        }

        //If not found return null
        return NULL;
    }

    //Return current size
    int sizeofMap()
    {
        return size;
    }

    //Return true if size is 0
    bool isEmpty()
    {
        return size == 0;
    }

    //Function to display the stored key value pairs
    void display()
    {
        for(int i=0 ; i<capacity ; i++)
        {
            if(arr[i] != NULL && arr[i]->key != -1)
                cout << "key = " << arr[i]->key
                     <<"  value = "<< arr[i]->value << endl;
        }
    }
};

//Driver method to test map class
int main()
{
    HashMap<int, int> *h = new HashMap<int, int>;
    h->insertNode(1,1);
    h->insertNode(2,2);
    h->insertNode(2,3);
    h->display();
    cout << h->sizeofMap() <<endl;
    cout << h->deleteNode(2) << endl;
    cout << h->sizeofMap() <<endl;
    cout << h->isEmpty() << endl;
    cout << h->get(2);

    return 0;
```

```
}
```

Output –

```
key = 1 value = 1
key = 2 value = 3
2
3
1
0
0
```

Source

https://www.geeksforgeeks.org/implementing-hash-table-open-addressing-linear-probing-cpp/

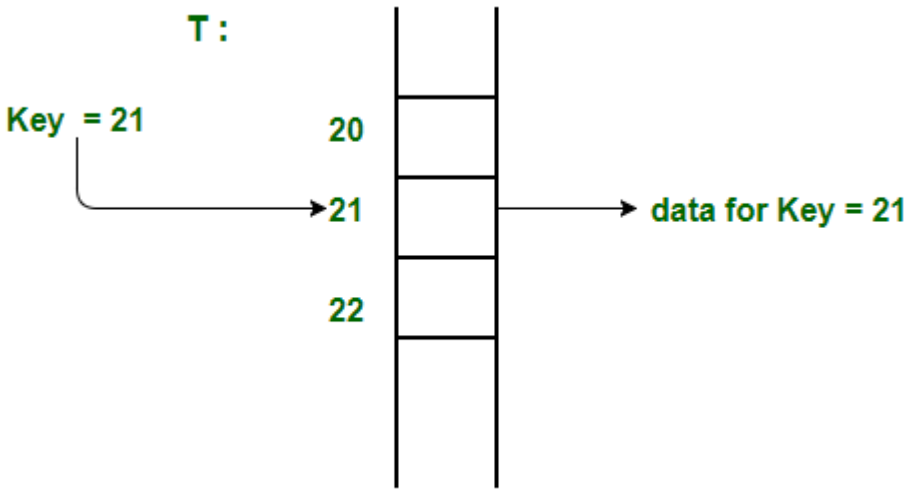
Chapter 160

Index Mapping (or Trivial Hashing) with negatives allowed

Index Mapping (or Trivial Hashing) with negatives allowed - GeeksforGeeks

Given a limited range array contains both positive and non positive numbers, i.e., elements are in range from -MAX to +MAX. Our task is to search if some number is present in the array or not in O(1) time.

Since range is limited, we can use index mapping (or trivial hashing). We use values as index in a big array. Therefore we can search and insert elements in O(1) time.



How to handle negative numbers?
The idea is to use a 2D array of size hash[MAX+1][2]

Algorithm:

```
Assign all the values of the hash matrix as 0.
Traverse the given array:
    If the element ele is non negative assign
        hash[ele][0] as 1.
    Else take the absolute value of ele and
         assign hash[ele][1] as 1.
```

To search any element x in the array.

- If X is non-negative check if hash[X][0] is 1 or not. If hash[X][0] is one then the number is present else not present.
- If X is negative take absolute vale of X and then check if hash[X][1] is 1 or not. If hash[X][1] is one then the number is present

Below is the implementation of the above idea.

```
 // CPP program to implement direct index mapping
// with negative values allowed.
#include <bits/stdc++.h>
using namespace std;
#define MAX 1000

// Since array is global, it is initialized as 0.
bool has[MAX + 1][2];

// searching if X is Present in the given array
// or not.
bool search(int X)
{
    if (X >= 0) {
        if (has[X][0] == 1)
            return true;
        else
            return false;
    }

    // if X is negative take the absolute
    // value of X.
    X = abs(X);
    if (has[X][1] == 1)
        return true;

    return false;
}

void insert(int a[], int n)
```

```
{
    for (int i = 0; i < n; i++) {
        if (a[i] >= 0)
            has[a[i]][0] = 1;
       else
            has[abs(a[i])][1] = 1;
    }
}

// Driver code
int main()
{
    int a[] = { -1, 9, -5, -8, -5, -2 };
    int n = sizeof(a)/sizeof(a[0]);
    insert(a, n);
    int X = -5;
    if (search(X) == true)
       cout << "Present";
    else
       cout << "Not Present";
    return 0;
}
```

Output:

```
Present
```

Improved By : fsociety_|_

Source

https://www.geeksforgeeks.org/index-mapping-or-trivial-hashing-with-negatives-allowed/

Chapter 161

Internal Working of HashMap in Java

Internal Working of HashMap in Java - GeeksforGeeks

In this article, we will see how hashmap's get and put method works internally. What operations are performed. How the hashing is done. How the value is fetched by key. How the key-value pair is stored.

As in previous article, HashMap contains an array of Node and Node can represent a class having following objects :

1. int hash
2. K key
3. V value
4. Node next

Now we will see how this works. First we will see the hashing process.

Hashing

Hashing is a process of converting an object into integer form by using the method hashCode(). Its necessary to write hashCode() method properly for better performance of HashMap. Here I am taking key of my own class so that I can override hashCode() method to show different scenarios. My Key class is

```
//custom Key class to override hashCode()
// and equals() method
class Key
{
  String key;
  Key(String key)
  {
```

```
    this.key = key;
  }

  @Override
  public int hashCode()
  {
     return (int)key.charAt(0);
  }

  @Override
  public boolean equals(Object obj)
  {
    return key.equals((String)obj);
  }
}
```

Here overrided hashCode() method returns the first character's ASCII value as hash code. So whenever the first character of key is same, the hash code will be same. You should not approach this criteria in your program. It is just for demo purpose. As HashMap also allows null key, so hash code of null will always be 0.

hashCode() method

hashCode() method is used to get the hash Code of an object. hashCode() method of object class returns the memory reference of object in integer form. Definition of hashCode() method is public native hashCode(). It indicates the implementation of hashCode() is native because there is not any direct method in java to fetch the reference of object. It is possible to provide your own implementation of hashCode().
In HashMap, hashCode() is used to calculate the bucket and therefore calculate the index.

equals() method

equals method is used to check that 2 objects are equal or not. This method is provided by Object class. You can override this in your class to provide your own implementation.
HashMap uses equals() to compare the key whether the are equal or not. If equals() method return true, they are equal otherwise not equal.

Buckets

A bucket is one element of HashMap array. It is used to store nodes. Two or more nodes can have the same bucket. In that case link list structure is used to connect the nodes. Buckets are different in capacity. A relation between bucket and capacity is as follows:

```
capacity = number of buckets * load factor
```

A single bucket can have more than one nodes, it depends on hashCode() method. The better your hashCode() method is, the better your buckets will be utilized.

Index Calculation in Hashmap

Hash code of key may be large enough to create an array. hash code generated may be in the range of integer and if we create arrays for such a range, then it will easily cause

outOfMemoryException. So we generate index to minimize the size of array. Basically following operation is performed to calculate index.

```
index = hashCode(key) & (n-1).
```

where n is number of buckets or the size of array. In our example, I will consider n as default size that is 16.

- **Initially Empty hashMap:** Here, the hashmap is size is taken as 16.

  ```
  HashMap map = new HashMap();
  ```


  HashMap :

- **Inserting Key-Value Pair:** Putting one key-value pair in above HashMap

  ```
  map.put(new Key("vishal"), 20);
  ```

  Steps:

  1. Calculate hash code of Key {“vishal”}. It will be generated as 118.
  2. Calculate index by using index method it will be 6.
  3. Create a node object as :

     ```
     {
       int hash = 118

       // {"vishal"} is not a string but
       // an object of class Key
       Key key = {"vishal"}

       Integer value = 20
       Node next = null
     }
     ```

  4. Place this object at index 6, if no other object is presented there.

  Now HashMap becomes :

- **Inserting another Key-Value Pair:** Now, putting other pair that is,

  ```
  map.put(new Key("sachin"), 30);
  ```

Steps:

1. Calculate hashCode of Key {“sachin”}. It will be generated as 115.
2. Calculate index by using index method it will be 3.
3. Create a node object as :

```
{
  int hash = 115
  Key key = {"sachin"}
  Integer value = 30
  Node next = null
}
```

- **In Case of collision:** Now, putting another pair that is,

```
map.put(new Key("vaibhav"), 40);
```

Steps:

1. Calculate hash code of Key {“vaibhav”}. It will be generated as 118.
2. Calculate index by using index method it will be 6.
3. Create a node object as :

```
 {
  int hash = 118
  Key key = {"vaibhav"}
  Integer value = 40
  Node next = null
}
```

4. Place this object at index 6 if no other object is presented there.
5. In this case a node object is **found at the index 6** – this is a case of collision.
6. In that case, check via hashCode() and equals() method that if both the keys are same.
7. If keys are same, replace the value with current value.
8. Otherwise connect this node object to the previous node object via linked list and both are stored at index 6.

Now HashMap becomes : 

Using get method()

Now lets try some get method to get a value. get(K key) method is used to get a value by its key. If you don’t know the key then it is not possible to fetch a value.

- **Fetch the data for key sachin:**

```
map.get(new Key("sachin"));
```

Steps:

1. Calculate hash code of Key {"sachin"}. It will be generated as 115.
2. Calculate index by using index method it will be 3.
3. Go to index 3 of array and compare first element's key with given key. If both are equals then return the value, otherwise check for next element if it exists.
4. In our case it is found as first element and returned value is 30.

- Fetch the data for key vaibahv:

```
map.get(new Key("vaibhav"));
```

Steps:

1. Calculate hash code of Key {"vaibhav"}. It will be generated as 118.
2. Calculate index by using index method it will be 6.
3. Go to index 6 of array and compare first element's key with given key. If both are equals then return the value, otherwise check for next element if it exists.
4. In our case it is not found as first element and next of node object is not null.
5. If next of node is null then return null.
6. If next of node is not null traverse to the second element and repeat the process 3 until key is not found or next is not null.

```
hashCode for key: vishal = 118
hashCode for key: sachin = 115
hashCode for key: vaibhav = 118

hashCode for key: sachin = 115
Value for key sachin: 30
hashCode for key: vaibhav = 118
Value for key vaibhav: 40
```

HashMap Changes in Java 8

As we know now that in case of hash collision entry objects are stored as a node in alinked-list and equals() method is used to compare keys. That comparison to find the correct key with in a linked-list is a linear operation so in a worst case scenario the complexity becomes O(n).
To address this issue, Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached. Which means HashMap starts with storing Entry objects in linked list but after the number of items in a hash becomes larger than a certain threshold, the hash will change from using a linked list to a balanced tree, which will improve the worst case performance from O(n) to O(log n).

Important Points

1. Time complexity is almost constant for put and get method until rehashing is not done.
2. In case of collision, i.e. index of two or more nodes are same, nodes are joined by link list i.e. second node is referenced by first node and third by second and so on.
3. If key given already exist in HashMap, the value is replaced with new value.
4. hash code of null key is 0.
5. When getting an object with its key, the linked list is traversed until the key matches or null is found on next field.

Improved By : ShubhamDeshmukh

Source

https://www.geeksforgeeks.org/internal-working-of-hashmap-java/

Chapter 162

Inverted Index

Inverted Index - GeeksforGeeks

An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents. In simple words, it is a hashmap like data structure that directs you from a word to a document or a web page.

There are two types of inverted indexes: A **record-level inverted index** contains a list of references to documents for each word. A **word-level inverted index** additionally contains the positions of each word within a document. The latter form offers more functionality, but needs more processing power and space to be created.
Suppose we want to search the texts "hello everyone, " "this article is based on inverted index, " "which is hashmap like data structure". If we index by (text, word within the text), the index with location in text is:

```
hello               (1, 1)
everyone            (1, 2)
this                (2, 1)
article             (2, 2)
is                  (2, 3); (3, 2)
based               (2, 4)
on                  (2, 5)
inverted            (2, 6)
index               (2, 7)
which               (3, 1)
hashmap             (3, 3)
like                (3, 4)
data                (3, 5)
structure           (3, 6)
```

The word "hello" is in document 1 ("hello everyone") starting at word 1, so has an entry (1, 1) and word "is" is in document 2 and 3 at '3rd' and '2nd' positions respectively (here

position is based on word).
The index may have weights, frequencies, or other indicators.

Steps to build an inverted index:

- **Fetch the Document**
  Removing of Stop Words: Stop words are most occuring and useless words in document like “I”, “the”, “we”, “is”, “an”.
- **Stemming of Root Word**
  Whenever I want to search for “cat”, I want to see a document that has information about it. But the word present in the document is called “cats” or “catty” instead of “cat”. To relate the both words, I’ll chop some part of each and every word I read so that I could get the “root word”. There are standard tools for performing this like “Porter’s Stemmer”.
- **Record Document IDs**
  If word is already present add reference of document to index else create new entry. Add additional information like frequency of word, location of word etc.

Example:

Words	Document
ant	doc1
demo	doc2
world	doc1, doc2

Advantage of Inverted Index are:

- Inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database.
- It is easy to develop.
- It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

Inverted Index also has disadvantage:

- Large storage overhead and high maintaenance costs on update, delete and insert.

Difference between Inverted Index and Forward Index

Source

https://www.geeksforgeeks.org/inverted-index/

Chapter 163

LFU (Least Frequently Used) Cache Implementation

LFU (Least Frequently Used) Cache Implementation - GeeksforGeeks

Least Frequently Used (LFU) is a caching algorithm in which the least frequently used cache block is removed whenever the cache is overflowed. In LFU we check the old page as well as the frequency of that page and if the frequency of the page is larger than the old page we cannot remove it and if all the old pages are having same frequency then take last i.e FIFO method for that and remove that page.

Min-heap data structure is a good option to implement this algorithm, as it handles insertion, deletion, and update in logarithmic time complexity. A tie can be resolved by removing the least recently used cache block. The following two containers have been used to solve the problem:

- A vector of integer pairs has been used to represent the cache, where each pair consists of the block number and the number of times it has been used. The vector is ordered in the form of a min-heap, which allows us to access the least frequently used block in constant time.
- A hashmap has been used to store the indices of the cache blocks which allows searching in constant time.

Below is the implementation of the above approach:

```
 // C++ program for LFU cache implementation
#include <bits/stdc++.h>
using namespace std;

// Generic function to swap two pairs
void swap(pair<int, int>& a, pair<int, int>& b)
{
```

```
    pair<int, int> temp = a;
    a = b;
    b = temp;
}

// Returns the index of the parent node
inline int parent(int i)
{
    return (i - 1) / 2;
}

// Returns the index of the left child node
inline int left(int i)
{
    return 2 * i + 1;
}

// Returns the index of the right child node
inline int right(int i)
{
    return 2 * i + 2;
}

// Self made heap tp Rearranges
//  the nodes in order to maintain the heap property
void heapify(vector<pair<int, int> >& v,
             unordered_map<int, int>& m, int i, int n)
{
    int l = left(i), r = right(i), minim;
    if (l < n)
        minim = ((v[i].second < v[l].second) ? i : l);
    else
        minim = i;
    if (r < n)
        minim = ((v[minim].second < v[r].second) ? minim : r);
    if (minim != i) {
        m[v[minim].first] = i;
        m[v[i].first] = minim;
        swap(v[minim], v[i]);
        heapify(v, m, minim, n);
    }
}

// Function to Increment the frequency
// of a node and rearranges the heap
void increment(vector<pair<int, int> >& v,
               unordered_map<int, int>& m, int i, int n)
{
```

```
    ++v[i].second;
    heapify(v, m, i, n);
}

// Function to Insert a new node in the heap
void insert(vector<pair<int, int> >& v,
            unordered_map<int, int>& m, int value, int& n)
{

    if (n == v.size()) {
        m.erase(v[0].first);
        cout << "Cache block " << v[0].first
                              << " removed.\n";
        v[0] = v[--n];
        heapify(v, m, 0, n);
    }
    v[n++] = make_pair(value, 1);
    m.insert(make_pair(value, n - 1));
    int i = n - 1;

    // Insert a node in the heap by swapping elements
    while (i && v[parent(i)].second > v[i].second) {
        m[v[i].first] = parent(i);
        m[v[parent(i)].first] = i;
        swap(v[i], v[parent(i)]);
        i = parent(i);
    }
    cout << "Cache block " << value << " inserted.\n";
}

// Function to refer to the block value in the cache
void refer(vector<pair<int, int> >& cache, unordered_map<int,
                    int>& indices, int value, int& cache_size)
{
    if (indices.find(value) == indices.end())
        insert(cache, indices, value, cache_size);
    else
        increment(cache, indices, indices[value], cache_size);
}

// Driver Code
int main()
{
    int cache_max_size = 4, cache_size = 0;
    vector<pair<int, int> > cache(cache_max_size);
    unordered_map<int, int> indices;
    refer(cache, indices, 1, cache_size);
    refer(cache, indices, 2, cache_size);
```

```
    refer(cache, indices, 1, cache_size);
    refer(cache, indices, 3, cache_size);
    refer(cache, indices, 2, cache_size);
    refer(cache, indices, 4, cache_size);
    refer(cache, indices, 5, cache_size);
    return 0;
}
```

Output:

```
Cache block 1 inserted.
Cache block 2 inserted.
Cache block 3 inserted.
Cache block 4 inserted.
Cache block 3 removed.
Cache block 5 inserted.
```

Source

https://www.geeksforgeeks.org/lfu-least-frequently-used-cache-implementation/

Chapter 164

Largest increasing subsequence of consecutive integers

Largest increasing subsequence of consecutive integers - GeeksforGeeks

Given an array of n positive integers. We need to find the largest increasing sequence of consecutive positive integers.

Examples:

```
Input : arr[] = {5, 7, 6, 7, 8}
Output : Size of LIS = 4
         LIS = 5, 6, 7, 8

Input : arr[] = {5, 7, 8, 7, 5}
Output : Size of LIS = 2
         LIS = 7, 8
```

This problem can be solved easily by the concept ofLIS where each next greater element differ from earlier one by 1. But this will take O(n^2) time complexity.

With the use of hashing we can finding the size of longest increasing sequence with consecutive integers in time complexity of O(n).

We create a hash table.. Now for each element arr[i], we perform hash[arr[i]] = hash[arr[i] – 1] + 1. So, for every element we know longest consecutive increasing subsequence ending with it. Finally we return maximum value from hash table.

```
 // C++ implementation of longest continuous increasing
// subsequence
#include <bits/stdc++.h>
using namespace std;
```

```
// Function for LIS
int findLIS(int A[], int n)
{
    unordered_map<int, int> hash;

    // Initialize result
    int LIS_size = 1;
    int LIS_index = 0;

    hash[A[0]] = 1;

    // iterate through array and find
    // end index of LIS and its Size
    for (int i = 1; i < n; i++) {
        hash[A[i]] = hash[A[i] - 1] + 1;
        if (LIS_size < hash[A[i]]) {
            LIS_size = hash[A[i]];
            LIS_index = A[i];
        }
    }

    // print LIS size
    cout << "LIS_size = " << LIS_size << "\n";

    // print LIS after setting start element
    cout << "LIS : ";
    int start = LIS_index - LIS_size + 1;
    while (start <= LIS_index) {
        cout << start << " ";
        start++;
    }
}

// driver
int main()
{
    int A[] = { 2, 5, 3, 7, 4, 8, 5, 13, 6 };
    int n = sizeof(A) / sizeof(A[0]);
    findLIS(A, n);
    return 0;
}
```

Output:

```
LIS_size = 5
LIS : 2 3 4 5 6
```

Source

https://www.geeksforgeeks.org/largest-increasing-subsequence-of-consecutive-integers/

Chapter 165

Largest palindromic number by permuting digits

Largest palindromic number by permuting digits - GeeksforGeeks

Given N(very large), task if to print the largest palindromic number obtained by permuting the digits of N. If it is not possible to make a palindromic number, then print an appropriate message.

Examples :

```
Input : 313551
Output : 531135
Explanations : 531135 is the largest number
which is a palindrome, 135531, 315513 and other
numbers can also be formed but we need the highest
of all of the palindromes.

Input : 331
Output : 313

Input : 3444
Output : Pallindrome cannot be formed
```

Naive Approach : The **naive** approach will be to try all the permutations possible, and print the largest of such combinations which is a palindrome.

Efficient Approach : An efficient approach will be to use **Greedy algorithm.** Since the number is large, store the number in a string. Store the count of occurrences of every digit in the given number in a map. Check if it is possible to form a palindrome or not. If the digits of the given number can be rearranged to form a palindrome, then apply the greedy approach to obtain the number. Check for the occurrence of every digit (9 to 0), and place

every available digit at front and back. **Initially, the front pointer will be at index 0, as the largest digit will be placed at first to make the number a large one.** With every step, move the front pointer 1 position ahead. If the digit occurs an odd number of times, then place **one digit in the middle and rest of the even number of digits at front and back**. Keep repeating the process **(map[digit]/2)** number of times for a single digit. After placing a particular digit which occurs an even number of times at the front and back, move the front pointer one step ahead. The placing is done till map[digit] is 0. The char array will have the largest palindromic number possible after completion of the placing of digits greedily.

In the worst case, the time complexity will be **O(10 * (length of string/2))**, in case the number consists of a same digit at every position.

Below is the implementation of the above idea :

```
 // CPP program to print the largest palindromic
// number by permuting digits of a number
#include <bits/stdc++.h>
using namespace std;

// function to check if a number can be
// permuted to form a palindrome number
bool possibility(unordered_map<int, int> m,
                 int length, string s)
{
    // counts the occurrence of number which is odd
    int countodd = 0;
    for (int i = 0; i < length; i++) {

        // if occurrence is odd
        if (m[s[i] - '0'] & 1)
            countodd++;

        // if number exceeds 1
        if (countodd > 1)
            return false;
    }

    return true;
}

// function to print the largest palindromic number
// by permuting digits of a number
void largestPalindrome(string s)
{

    // string length
    int l = s.length();
```

```
// map that marks the occurrence of a number
unordered_map<int, int> m;
for (int i = 0; i < l; i++)
    m[s[i] - '0']++;

// check the possibility of a palindromic number
if (possibility(m, l, s) == false) {
    cout << "Palindrome cannot be formed";
    return;
}

// string array that stores the largest
// permuted palindromic number
char largest[l];

// pointer of front
int front = 0;

// greedily start from 9 to 0 and place the
// greater number in front and odd in the
// middle
for (int i = 9; i >= 0; i--) {

    // if the occurrence of number is odd
    if (m[i] & 1) {

        // place one odd occurring number
        // in the middle
        largest[l / 2] = char(i + 48);

        // decrease the count
        m[i]--;

        // place the rest of numbers greedily
        while (m[i] > 0) {
            largest[front] = char(i + 48);
            largest[l - front - 1] = char(i + 48);
            m[i] -= 2;
            front++;
        }
    }
    else {

        // if all numbers occur even times,
        // then place greedily
        while (m[i] > 0) {

            // place greedily at front
```

```
                largest[front] = char(i + 48);
                largest[l - front - 1] = char(i + 48);

                // 2 numbers are placed, so decrease the count
                m[i] -= 2;

                // increase placing position
                front++;
            }
        }
    }

    // print the largest string thus formed
    for (int i = 0; i < l; i++)
        cout << largest[i];
}

// Driver Code
int main()
{
    string s = "313551";
    largestPalindrome(s);
    return 0;
}
```

Output:

```
531135
```

Time Complexity : O(n), where n is the length of string.

Source

https://www.geeksforgeeks.org/largest-palindromic-number-permuting-digits/

Chapter 166

Largest subarray with equal number of 0s and 1s

Largest subarray with equal number of 0s and 1s - GeeksforGeeks

Given an array containing only 0s and 1s, find the largest subarray which contain equal no of 0s and 1s. Expected time complexity is O(n).

Examples:

```
Input: arr[] = {1, 0, 1, 1, 1, 0, 0}
Output: 1 to 6 (Starting and Ending indexes of output subarray)

Input: arr[] = {1, 1, 1, 1}
Output: No such subarray

Input: arr[] = {0, 0, 1, 1, 0}
Output: 0 to 3 Or 1 to 4
```

Method 1 (Simple)
A simple method is to use two nested loops. The outer loop picks a starting point i. The inner loop considers all subarrays starting from i. If size of a subarray is greater than maximum size so far, then update the maximum size.
In the below code, 0s are considered as -1 and sum of all values from i to j is calculated. If sum becomes 0, then size of this subarray is compared with largest size so far.

C

```
 // A simple program to find the largest subarray
// with equal number of 0s and 1s
```

```
#include <stdio.h>

// This function Prints the starting and ending
// indexes of the largest subarray with equal
// number of 0s and 1s. Also returns the size
// of such subarray.

int findSubArray(int arr[], int n)
{
    int sum = 0;
    int maxsize = -1, startindex;

    // Pick a starting point as i

    for (int i = 0; i < n-1; i++)
    {
        sum = (arr[i] == 0)? -1 : 1;

        // Consider all subarrays starting from i

        for (int j = i+1; j < n; j++)
        {
            (arr[j] == 0)? (sum += -1): (sum += 1);

            // If this is a 0 sum subarray, then
            // compare it with maximum size subarray
            // calculated so far

            if (sum == 0 && maxsize < j-i+1)
            {
                maxsize = j - i + 1;
                startindex = i;
            }
        }
    }
    if (maxsize == -1)
        printf("No such subarray");
    else
        printf("%d to %d", startindex, startindex+maxsize-1);

    return maxsize;
}

/* Driver program to test above functions*/

int main()
{
    int arr[] =  {1, 0, 0, 1, 0, 1, 1};
```

```
    int size = sizeof(arr)/sizeof(arr[0]);

    findSubArray(arr, size);
    return 0;
}
```

Java

```
 class LargestSubArray
{

    // This function Prints the starting and ending
    // indexes of the largest subarray with equal
    // number of 0s and 1s. Also returns the size
    // of such subarray.

    int findSubArray(int arr[], int n)
    {
        int sum = 0;
        int maxsize = -1, startindex = 0;
        int endindex = 0;

        // Pick a starting point as i

        for (int i = 0; i < n - 1; i++)
        {
            sum = (arr[i] == 0) ? -1 : 1;

            // Consider all subarrays starting from i

            for (int j = i + 1; j < n; j++)
            {
                if(arr[j] == 0)
                    sum += -1;
                else
                    sum += 1;

                // If this is a 0 sum subarray, then
                // compare it with maximum size subarray
                // calculated so far

                if (sum == 0 && maxsize < j - i + 1)
                {
                    maxsize = j - i + 1;
                    startindex = i;
                }
            }
        }
```

```
        endindex = startindex+maxsize-1;
        if (maxsize == -1)
            System.out.println("No such subarray");
        else
            System.out.println(startindex+" to "+endindex);

        return maxsize;
    }

    /* Driver program to test the above functions */

    public static void main(String[] args)
    {
        LargestSubArray sub;
        sub = new LargestSubArray();
        int arr[] = {1, 0, 0, 1, 0, 1, 1};
        int size = arr.length;

        sub.findSubArray(arr, size);
    }
}
```

Python3

```
 # A simple program to find the largest subarray
# with equal number of 0s and 1s

# This function Prints the starting and ending
# indexes of the largest subarray with equal
# number of 0s and 1s. Also returns the size
# of such subarray.
def findSubArray(arr, n):

    sum = 0
    maxsize = -1

    # Pick a starting point as i

    for i in range(0, n-1):

        sum = -1 if(arr[i] == 0) else 1

        # Consider all subarrays starting from i

        for j in range(i + 1, n):

            sum = sum + (-1) if (arr[j] == 0) else sum + 1
```

```
            # If this is a 0 sum subarray, then
            # compare it with maximum size subarray
            # calculated so far

            if (sum == 0 and maxsize < j-i + 1):

                maxsize = j - i + 1
                startindex = i



    if (maxsize == -1):
        print("No such subarray");
    else:
        print(startindex, "to", startindex + maxsize-1);

    return maxsize

# Driver program to test above functions
arr = [1, 0, 0, 1, 0, 1, 1]
size = len(arr)
findSubArray(arr, size)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // A simple program to find the largest subarray
// with equal number of 0s and 1s
using System;

class GFG
{

    // This function Prints the starting and ending
    // indexes of the largest subarray with equal
    // number of 0s and 1s. Also returns the size
    // of such subarray.

    static int findSubArray(int []arr, int n)
    {
        int sum = 0;
        int maxsize = -1, startindex = 0;
        int endindex = 0;

        // Pick a starting point as i
        for (int i = 0; i < n - 1; i++)
        {
```

```
            sum = (arr[i] == 0) ? -1 : 1;

            // Consider all subarrays starting from i

            for (int j = i + 1; j < n; j++)
            {
                if(arr[j] == 0)
                    sum += -1;
                else
                    sum += 1;

                // If this is a 0 sum subarray, then
                // compare it with maximum size subarray
                // calculated so far

                if (sum == 0 && maxsize < j - i + 1)
                {
                    maxsize = j - i + 1;
                    startindex = i;
                }
            }
        }
        endindex = startindex+maxsize-1;
        if (maxsize == -1)
            Console.WriteLine("No such subarray");
        else
            Console.WriteLine(startindex+" to "+endindex);

        return maxsize;
    }

    // Driver program
    public static void Main()
    {

        int []arr = {1, 0, 0, 1, 0, 1, 1};
        int size = arr.Length;
        findSubArray(arr, size);
    }
}

// This code is contributed by Sam007
```

Output:

```
 0 to 5
```

Time Complexity: O(n^2)

Auxiliary Space: O(1)

Method 2 (Tricky)
Following is a solution that uses O(n) extra space and solves the problem in O(n) time complexity.
Let input array be arr[] of size n and maxsize be the size of output subarray.
1) Consider all 0 values as -1. The problem now reduces to find out the maximum length subarray with sum = 0.
2) Create a temporary array sumleft[] of size n. Store the sum of all elements from arr[0] to arr[i] in sumleft[i]. This can be done in O(n) time.
3) There are two cases, the output subarray may start from 0th index or may start from some other index. We will return the max of the values obtained by two cases.
4) To find the maximum length subarray starting from 0th index, scan the sumleft[] and find the maximum i where sumleft[i] = 0.
5) Now, we need to find the subarray where subarray sum is 0 and start index is not 0. This problem is equivalent to finding two indexes i & j in sumleft[] such that sumleft[i] = sumleft[j] and j-i is maximum. To solve this, we can create a hash table with size = max-min+1 where min is the minimum value in the sumleft[] and max is the maximum value in the sumleft[]. The idea is to hash the leftmost occurrences of all different values in sumleft[]. The size of hash is chosen as max-min+1 because there can be these many different possible values in sumleft[]. Initialize all values in hash as -1
6) To fill and use hash[], traverse sumleft[] from 0 to n-1. If a value is not present in hash[], then store its index in hash. If the value is present, then calculate the difference of current index of sumleft[] and previously stored value in hash[]. If this difference is more than maxsize, then update the maxsize.
7) To handle corner cases (all 1s and all 0s), we initialize maxsize as -1. If the maxsize remains -1, then print there is no such subarray.

C

```
 // A O(n) program to find the largest subarray
// with equal number of 0s and 1s

#include <stdio.h>
#include <stdlib.h>

// A utility function to get maximum of two
// integers

int max(int a, int b) { return a>b? a: b; }

// This function Prints the starting and ending
// indexes of the largest subarray with equal
// number of 0s and 1s. Also returns the size
// of such subarray.

int findSubArray(int arr[], int n)
{
    // variables to store result values
```

```
int maxsize = -1, startindex;

// Create an auxiliary array sunmleft[].
// sumleft[i] will be sum of array
// elements from arr[0] to arr[i]

int sumleft[n];

// For min and max values in sumleft[]

int min, max;
int i;

// Fill sumleft array and get min and max
// values in it.  Consider 0 values in arr[]
// as -1

sumleft[0] = ((arr[0] == 0)? -1: 1);
min = arr[0]; max = arr[0];
for (i=1; i<n; i++)
{
    sumleft[i] = sumleft[i-1] + ((arr[i] == 0)?
                 -1: 1);
    if (sumleft[i] < min)
        min = sumleft[i];
    if (sumleft[i] > max)
        max = sumleft[i];
}

// Now calculate the max value of j - i such
// that sumleft[i] = sumleft[j]. The idea is
// to create a hash table to store indexes of all
// visited values.
// If you see a value again, that it is a case of
// sumleft[i] = sumleft[j]. Check if this j-i is
// more than maxsize.
// The optimum size of hash will be max-min+1 as
// these many different values of sumleft[i] are
// possible. Since we use optimum size, we need
// to shift all values in sumleft[] by min before
// using them as an index in hash[].

int hash[max-min+1];

// Initialize hash table

for (i=0; i<max-min+1; i++)
```

```
        hash[i] = -1;

    for (i=0; i<n; i++)
    {
        // Case 1: when the subarray starts from
        //         index 0

        if (sumleft[i] == 0)
        {
           maxsize = i+1;
           startindex = 0;
        }

        // Case 2: fill hash table value. If already
        //         filled, then use it

        if (hash[sumleft[i]-min] == -1)
            hash[sumleft[i]-min] = i;
        else
        {
            if ((i - hash[sumleft[i]-min]) > maxsize)
            {
                maxsize = i - hash[sumleft[i]-min];
                startindex = hash[sumleft[i]-min] + 1;
            }
        }
    }
    if (maxsize == -1)
        printf("No such subarray");
    else
        printf("%d to %d", startindex, startindex+maxsize-1);

    return maxsize;
}

/* Driver program to test above functions */
int main()
{
    int arr[] =  {1, 0, 0, 1, 0, 1, 1};
    int size = sizeof(arr)/sizeof(arr[0]);

    findSubArray(arr, size);
    return 0;
}
```

C++/STL

```
 // C++ program to find largest subarray with equal number of
```

```
// 0's and 1's.

#include <bits/stdc++.h>
using namespace std;

// Returns largest subarray with equal number of 0s and 1s

int maxLen(int arr[], int n)
{
    // Creates an empty hashMap hM

    unordered_map<int, int> hM;

    int sum = 0;     // Initialize sum of elements
    int max_len = 0; // Initialize result
    int ending_index = -1;

    for (int i = 0; i < n; i++)
        arr[i] = (arr[i] == 0)? -1: 1;

    // Traverse through the given array

    for (int i = 0; i < n; i++)
    {
        // Add current element to sum

        sum += arr[i];

        // To handle sum=0 at last index

        if (sum == 0)
        {
            max_len = i + 1;
            ending_index = i;
        }

        // If this sum is seen before, then update max_len
        // if required

        if (hM.find(sum + n) != hM.end())
        {
            if (max_len < i - hM[sum + n])
            {
                max_len = i - hM[sum + n];
                ending_index = i;
            }
        }
        else // Else put this sum in hash table
```

```
            hM[sum + n] = i;
    }

    for (int i = 0; i < n; i++)
        arr[i] = (arr[i] == -1)? 0: 1;

    printf("%d to %d\n", ending_index-max_len+1, ending_index);

    return max_len;
}

// Driver method

int main()
{
    int arr[] = {1, 0, 0, 1, 0, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    maxLen(arr, n);
    return 0;
}

// This code is contributed by Aditya Goel
```

Java

```
 import java.util.HashMap;

class LargestSubArray1
{

    // Returns largest subarray with equal number of 0s and 1s

    int maxLen(int arr[], int n)
    {
        // Creates an empty hashMap hM

        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        int sum = 0;     // Initialize sum of elements
        int max_len = 0; // Initialize result
        int ending_index = -1;
        int start_index = 0;

        for (int i = 0; i < n; i++)
        {
            arr[i] = (arr[i] == 0) ? -1 : 1;
        }
```

```
    // Traverse through the given array

    for (int i = 0; i < n; i++)
    {
        // Add current element to sum

        sum += arr[i];

        // To handle sum=0 at last index

        if (sum == 0)
        {
            max_len = i + 1;
            ending_index = i;
        }

        // If this sum is seen before, then update max_len
        // if required

        if (hM.containsKey(sum + n))
        {
            if (max_len < i - hM.get(sum + n))
            {
                max_len = i - hM.get(sum + n);
                ending_index = i;
            }
        }
        else // Else put this sum in hash table
            hM.put(sum + n, i);
    }

    for (int i = 0; i < n; i++)
    {
        arr[i] = (arr[i] == -1) ? 0 : 1;
    }

    int end = ending_index - max_len + 1;
    System.out.println(end + " to " + ending_index);

    return max_len;
}

/* Driver program to test the above functions */

public static void main(String[] args)
{
    LargestSubArray1 sub = new LargestSubArray1();
```

```
        int arr[] = {1, 0, 0, 1, 0, 1, 1};
        int n = arr.length;

        sub.maxLen(arr, n);
    }
}

// This code has been by Mayank Jaiswal(mayank_24)
```

Output:

```
 0 to 5
```

Time Complexity: O(n)
Auxiliary Space: O(n)
Thanks to Aashish Barnwal for suggesting this solution.

Improved By : Vijay Akkaladevi

Source

https://www.geeksforgeeks.org/largest-subarray-with-equal-number-of-0s-and-1s/

Chapter 167

Largest subset whose all elements are Fibonacci numbers

Largest subset whose all elements are Fibonacci numbers - GeeksforGeeks

Given an array with positive number the task is that we find largest subset from array that contain elements which are Fibonacci numbers.

Asked in Facebook

Examples :

```
Input : arr[] = {1, 4, 3, 9, 10, 13, 7};
Output : subset[] = {1, 3, 13}
The output three numbers are Fibonacci
numbers.

Input  : arr[] = {0, 2, 8, 5, 2, 1, 4,
                  13, 23};
Output : subset[] = {0, 2, 8, 5, 2, 1,
                    13, 23}
```

A **simple solution** is to iterate through all elements of given array. For every number, check if it is Fibonacci or not. If yes, add it to the result.

Below is an **efficient solution** based on hashing.

1. Find max in the array
2. Generate Fibonacci numbers till the max and store it in hash table.
3. Traverse array again if the number is present in hash table then add it to the result.

C++

```
 // C++ program to find largest Fibonacci subset
#include<bits/stdc++.h>
using namespace std;

// Prints largest subset of an array whose
// all elements are fibonacci numbers
void findFibSubset(int arr[], int n)
{
    // Find maximum element in arr[]
    int max = *std::max_element(arr, arr+n);

    // Generate all Fibonacci numbers till
    // max and store them in hash.
    int a = 0, b = 1;
    unordered_set<int> hash;
    hash.insert(a);
    hash.insert(b);
    while (b < max)
    {
        int c = a + b;
        a = b;
        b = c;
        hash.insert(b);
    }

    // Npw iterate through all numbers and
    // quickly check for Fibonacci using
    // hash.
    for (int i=0; i<n; i++)
        if (hash.find(arr[i]) != hash.end())
            printf("%d ", arr[i]);
}

// Driver code
int main()
{
    int arr[] = {4, 2, 8, 5, 20, 1, 40, 13, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    findFibSubset(arr, n);
    return 0;
}
```

Java

```
 // Java program to find
// largest Fibonacci subset
import java.util.*;
```

```
class GFG
{
    // Prints largest subset of an array whose
    // all elements are fibonacci numbers
    public static void findFibSubset(Integer[] x)
    {
        Integer max = Collections.max(Arrays.asList(x));
        List<Integer> fib = new ArrayList<Integer>();
        List<Integer> result = new ArrayList<Integer>();

        // Generate all Fibonacci numbers
        // till max and store them
        Integer a = 0;
        Integer b = 1;
        while (b < max){
            Integer c = a + b;
            a=b;
            b=c;
            fib.add(c);
        }

        // Now iterate through all numbers and
        // quickly check for Fibonacci
        for (Integer i = 0; i < x.length; i++){
        if(fib.contains(x[i])){
            result.add(x[i]);
        }
        }
        System.out.println(result);
    }

    // Driver code
    public static void main(String args[])
    {
        Integer[] a = {4, 2, 8, 5, 20, 1, 40, 13, 23};
        findFibSubset(a);
    }
}

// This code is contributed by prag93
```

Output:

```
2 8 5 1 13
```

Reference :
https://www.careercup.com/question?id=5154130839470080

Improved By : prag93

Source

https://www.geeksforgeeks.org/largest-subset-whose-all-elements-are-fibonacci-numbers/

Chapter 168

Last seen array element (last appearance is earliest)

Last seen array element (last appearance is earliest) - GeeksforGeeks

Given an array that might contain duplicates, find the element whose last appearance is latest.

Examples:

```
Input :  arr[] = {10, 30, 20, 10, 20}
Output : 30
Explanation: Below are indexes of last
appearances of all elements (0 based indexes)
10 last occurs at index 3
30 last occurs at index 1
20 last occurs at index 2
The element whose last appearance earliest
is 30.

Input : arr[] = {20, 10, 20, 20, 40, 10}
Output : 20
Explanation:
Explanation: Below are indexes of last
appearances of all elements (0 based indexes)
20 last occurs at index 2
10 last occurs at index 5
40 last occurs at index 4
The element whose last appearance earliest
is 20.
```

A **naive approach** is to take every element and iterate and compare its last occurrence with other elements. This requires two nested loops, and will take O(n*n) time.

An **efficient approach** is to use hashing. We store every elements index where it last occurred, and then iterate through all the possible elements and print the element with the least least index stored occurrence, as that will be the one which was last seen while traversing from left to right.

```
 // CPP program to find last seen element in
// an array.
#include <bits/stdc++.h>
using namespace std;

// Returns last seen element in arr[]
int lastSeenElement(int a[], int n)
{
    // Store last occurrence index of
    // every element
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++)
        hash[a[i]] = i;

    // Find an element in hash with minimum
    // index value
    int res_ind = INT_MAX, res;
    for (auto x : hash)
    {
       if (x.second < res_ind)
       {
            res_ind = x.second;
            res = x.first;
       }
    }

    return res;
}

// driver program
int main()
{
    int a[] = { 2, 1, 2, 2, 4, 1 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << lastSeenElement(a, n);
    return 0;
}
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/last-seen-array-element-last-appearance-earliest/

Chapter 169

Length of longest strict bitonic subsequence

Length of longest strict bitonic subsequence - GeeksforGeeks

Given an array arr[] containing n integers. The problem is to find the length of the longest strict bitonic subsequence. A subsequence is called **strict** bitonic if it is first increasing and then decreasing with the condition that in both the increasing and decreasing parts the absolute difference between adjacents is 1 only. A sequence, sorted in increasing order is considered Bitonic with the decreasing part as empty. Similarly, decreasing order sequence is considered Bitonic with the increasing part as empty.

Examples:

```
Input : arr[] = {1, 5, 2, 3, 4, 5, 3, 2}
Output : 6
The Longest Strict Bitonic Subsequence is:
{1, 2, 3, 4, 3, 2}.

Input : arr[] = {1, 2, 5, 3, 6, 7, 4, 6, 5}
Output : 5
```

Method 1: The problem could be solved using the concept of finding the longest bitonic subsequence. The only condition that needs to be maintained is that the adjacents should have a difference of 1 only. It has a time complexity of $O(n^2)$.

Method 2 (Efficient Approach): The idea is to create two hash maps **inc** and **dcr** having tuples in the form **(ele, len)**, where **len** denotes the length of the longest increasing subsequence ending with the element **ele** in map **inc** and length of the longest decreasing subsequence starting with element **ele** in map **dcr** respectively. Also create two arrays **len_inc**[] and **len_dcr**[] where **len_inc**[**i**] represents the length of the largest increasing subsequence ending with element **arr**[**i**] and **len_dcr**[**i**] represents the length of the largest

decreasing subsequence starting with element **arr[i]**. Now, for each element **arr[i]** we can find the length of the value **(arr[i]-1)** if it exists in the hash table **inc**. Let this value be **v (initially v will be 0)**. Now, the length of longest increasing subsequence ending with **arr[i]** would be **v+1**. Update this length along with the element arr[i] in the hash table **inc** and in the array **len_inc**[] at respective index **i**. Now, traversing the array from right to left we can similarly fill the hash table **dcr** and array **len_dcr**[] for longest decreasing subsequence. Finally, for each element arr[i] we calculate **(len_inc[i] + len_dcr[i] − 1)** and return the maximum value.

Note: Here increasing and decreasing subsequences only mean that the difference between adjacent elements is 1 only.

```
 // C++ implementation to find length of longest
// strict bitonic subsequence
#include <bits/stdc++.h>
using namespace std;

// function to find length of longest
// strict bitonic subsequence
int longLenStrictBitonicSub(int arr[], int n)
{
    // hash table to map the array element with the
    // length of the longest subsequence of which
    // it is a part of and is the last/first element of
    // that subsequence
    unordered_map<int, int> inc, dcr;

    // arrays to store the length of increasing and
    // decreasing subsequences which end at them
    // or start from them
    int len_inc[n], len_dcr[n];

    // to store the length of longest strict
    // bitonic subsequence
    int longLen = 0;

    // traverse the array elements
    // from left to right
    for (int i=0; i<n; i++)
    {
        // initialize current length
        // for element arr[i] as 0
        int len = 0;

        // if 'arr[i]-1' is in 'inc'
        if (inc.find(arr[i]-1) != inc.end())
            len = inc[arr[i]-1];

        // update arr[i] subsequence length in 'inc'
```

```
        // and in len_inc[]
        inc[arr[i]] = len_inc[i] = len + 1;
    }

    // traverse the array elements
    // from right to left
    for (int i=n-1; i>=0; i--)
    {
        // initialize current length
        // for element arr[i] as 0
        int len = 0;

        // if 'arr[i]-1' is in 'dcr'
        if (dcr.find(arr[i]-1) != dcr.end())
            len = dcr[arr[i]-1];

        // update arr[i] subsequence length in 'dcr'
        // and in len_dcr[]
        dcr[arr[i]] = len_dcr[i] = len + 1;
    }

    // calculating the length of all the strict
    // bitonic subsequence
    for (int i=0; i<n; i++)
        if (longLen < (len_inc[i] + len_dcr[i] - 1))
            longLen = len_inc[i] + len_dcr[i] - 1;

    // required longest length strict
    // bitonic subsequence
    return longLen;
}

// Driver program to test above
int main()
{
    int arr[] = {1, 5, 2, 3, 4, 5, 3, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Longest length strict bitonic subsequence = "
         << longLenStrictBitonicSub(arr, n);
    return 0;
}
```

Output:

```
Longest length strict bitonic subsequence = 6
```

Time Complexity: O(n).

Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/length-longest-strict-bitonic-subsequence/

Chapter 170

Length of the largest subarray with contiguous elements | Set 2

Length of the largest subarray with contiguous elements | Set 2 - GeeksforGeeks

Given an array of integers, find length of the longest subarray which contains numbers that can be arranged in a continuous sequence.

In theprevious post, we have discussed a solution that assumes that elements in given array are distinct. Here we discuss a solution that works even if the input array has duplicates.

Examples:

```
Input:  arr[] = {10, 12, 11};
Output: Length of the longest contiguous subarray is 3

Input:  arr[] = {10, 12, 12, 10, 10, 11, 10};
Output: Length of the longest contiguous subarray is 2
```

The idea is similar to previous post. In the previous post, we checked whether maximum value minus minimum value is equal to ending index minus starting index or not. Since duplicate elements are allowed, we also need to check if the subarray contains duplicate elements or not. For example, the array {12, 14, 12} follows the first property, but numbers in it are not contiguous elements.
To check duplicate elements in a subarray, we create a hash set for every subarray and if we find an element already in hash, we don't consider the current subarray.

Following is the implementation of the above idea.

C++

```
/* CPP program to find length of the largest
```

```
subarray which has all contiguous elements */
#include<bits/stdc++.h>
using namespace std;

// This function prints all distinct elements
int findLength(int arr[], int n)
{
    int max_len = 1; // Inialize result

    // One by one fix the starting points
    for (int i=0; i<n-1; i++)
    {
        // Create an empty hash set and
        // add i'th element to it.
        set<int> myset;
        myset.insert(arr[i]);

        // Initialize max and min in
        // current subarray
        int mn = arr[i], mx = arr[i];

        // One by one fix ending points
        for (int j=i+1; j<n; j++)
        {
            // If current element is already
            // in hash set, then this subarray
            // cannot contain contiguous elements
            if (myset.find(arr[j]) != myset.end())
                break;

            // Else add current element to hash
            // set and update min, max if required.
            myset.insert(arr[j]);
            mn = min(mn, arr[j]);
            mx = max(mx, arr[j]);

            // We have already checked for
            // duplicates, now check for other
            // property and update max_len
            // if needed
            if (mx - mn == j - i)
                max_len = max(max_len, mx - mn + 1);
        }
    }
    return max_len; // Return result
}

// Driver method to test above method
```

```
int main ()
{
    int arr[] = {10, 12, 12, 10, 10, 11, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Length of the longest contiguous"
         << " subarray is " << findLength(arr, n);
}

// This article is contributed by Chhavi
```

Java

```
 /* Java program to find length of the largest subarray which has
   all contiguous elements */
import java.util.*;

class Main
{
    // This function prints all distinct elements
    static int findLength(int arr[])
    {
        int n = arr.length;
        int max_len = 1; // Inialize result

        // One by one fix the starting points
        for (int i=0; i<n-1; i++)
        {
            // Create an empty hash set and add i'th element
            // to it.
            HashSet<Integer> set = new HashSet<>();
            set.add(arr[i]);

            // Initialize max and min in current subarray
            int mn = arr[i], mx = arr[i];

            // One by one fix ending points
            for (int j=i+1; j<n; j++)
            {
                // If current element is already in hash set, then
                // this subarray cannot contain contiguous elements
                if (set.contains(arr[j]))
                    break;

                // Else add curremt element to hash set and update
                // min, max if required.
                set.add(arr[j]);
                mn = Math.min(mn, arr[j]);
                mx = Math.max(mx, arr[j]);
```

```
                // We have already cheched for duplicates, now check
                // for other property and update max_len if needed
                if (mx-mn == j-i)
                    max_len = Math.max(max_len, mx-mn+1);
            }
        }
        return max_len; // Return result
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
       int arr[] =  {10, 12, 12, 10, 10, 11, 10};
       System.out.println("Length of the longest contiguous subarray is " +
                          findLength(arr));
    }
}
```

C#

```
 /* C# program to find length of the largest
subarray which has all contiguous elements */
using System;
using System.Collections.Generic;

class GFG {

    // This function prints all distinct
    // elements
    static int findLength(int []arr)
    {
        int n = arr.Length;
        int max_len = 1; // Inialize result

        // One by one fix the starting points
        for (int i = 0; i < n-1; i++)
        {

            // Create an empty hash set and
            // add i'th element to it.
            HashSet<int> set = new HashSet<int>();
            set.Add(arr[i]);

            // Initialize max and min in current
            // subarray
            int mn = arr[i], mx = arr[i];
```

```
            // One by one fix ending points
            for (int j = i+1; j < n; j++)
            {

                // If current element is already
                // in hash set, then this subarray
                // cannot contain contiguous
                // elements
                if (set.Contains(arr[j]))
                    break;

                // Else add curremt element to
                // hash set and update min,
                // max if required.
                set.Add(arr[j]);
                mn = Math.Min(mn, arr[j]);
                mx = Math.Max(mx, arr[j]);

                // We have already cheched for
                // duplicates, now check for
                // other property and update
                // max_len if needed
                if (mx-mn == j-i)
                    max_len = Math.Max(max_len,
                                  mx - mn + 1);
            }
        }

        return max_len; // Return result
    }

    // Driver function
    public static void Main()
    {
        int []arr = {10, 12, 12, 10, 10, 11, 10};
        Console.WriteLine("Length of the longest"
                  + " contiguous subarray is " +
                              findLength(arr));
    }
}

// This code is contributed by Sam007
```

Output:

```
Length of the longest contiguous subarray is 2
```

Time complexity of the above solution is $O(n^2)$ under the assumption that hash set opera-

tions like add() and contains() work in O(1) time.

This article is contributed by **Arjun**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007

Source

https://www.geeksforgeeks.org/length-largest-subarray-contiguous-elements-set-2/

Chapter 171

Length of the longest substring with equal 1s and 0s

Length of the longest substring with equal 1s and 0s - GeeksforGeeks

Given a binary string. We need to find the length of longest balanced sub string. A sub string is balanced if it contains equal number of 0 and 1.

Examples:

```
Input : input = 110101010
Output : Length of longest balanced
         sub string = 8

Input : input = 0000
Output : Length of longest balanced
         sub string = 0
```

A **simple solution** is to use two nested loops to generate every substring. And a third loop to count number of 0s and 1s in current substring. Time complexity of this would be $O(n^3)$

An **efficient solution** is to use hashing.
1) Traverse string and keep track of counts of 1s and 0s as count_1 and count_0 respectively.
2) See if current difference between two counts has appeared before (We use hashing to store all differences and first index where a difference appears). If yes, then substring from previous appearance and current index has same number of 0s and 1s.

C++

```
 // CPP for finding length of longest balanced
// substring
#include<bits/stdc++.h>
using namespace std;
```

```
// Returns length of the longest substring
// with equal number of zeros and ones.
int stringLen(string str)
{
    // Create a map to store differences
    // between counts of 1s and 0s.
    map<int, int> m;

    // Initially difference is 0.
    m[0] = -1;

    int count_0 = 0, count_1 = 0;
    int res = 0;
    for (int i=0; i<str.size(); i++)
    {
        // Keeping track of counts of
        // 0s and 1s.
        if (str[i] == '0')
            count_0++;
        else
            count_1++;

        // If difference between current counts
        // already exists, then substring between
        // previous and current index has same
        // no. of 0s and 1s. Update result if this
        // substring is more than current result.
        if (m.find(count_1 - count_0) != m.end())
            res = max(res, i - m[count_1 - count_0]);

        // If current difference is seen first time.
        else
            m[count_1 - count_0] = i;
    }

    return res;
}

// driver function
int main()
{
    string str = "101001000";
    cout << "Length of longest balanced"
            " sub string = ";
    cout << stringLen(str);
    return 0;
}
```

Python3

```
 # Python3 code for finding length of
# longest balanced substring

# Returns length of the longest substring
# with equal number of zeros and ones.
def stringLen( str ):

    # Create a python dictionary to store
    # differences between counts of 1s and 0s.
    m = dict()

    # Initially difference is 0.
    m[0] = -1

    count_0 = 0
    count_1 = 0
    res = 0
    for i in range(len(str)):

        # Keeping track of counts of
        # 0s and 1s.
        if str[i] == '0':
            count_0 += 1
        else:
            count_1 += 1

        # If difference between current
        # counts already exists, then
        # substring between previous and
        # current index has same no. of
        # 0s and 1s. Update result if
        # this substring is more than
        # current result.
        if m.get(count_1 - count_0):
            res = max(res, i - m[count_1 - count_0])

        # If current difference is
        # seen first time.
        else:
            m[count_1 - count_0] = i
    return res

# driver code
str = "101001000"
print("Length of longest balanced"
     " sub string = ",stringLen(str))
```

```
# This code is contributed by "Sharad_Bhardwaj"
```

Output:

```
Length of longest balanced sub string = 6
```

Time Complexity : O(n)

Extended Problem : Largest subarray with equal number of 0s and 1s

Source

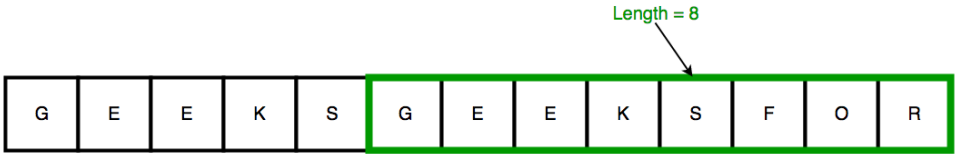
https://www.geeksforgeeks.org/length-of-the-longest-substring-with-equal-1s-and-0s/

Chapter 172

Length of the smallest sub-string consisting of maximum distinct characters

Length of the smallest sub-string consisting of maximum distinct characters - GeeksforGeeks

Given a string of length N, find the length of the smallest sub-string consisting of maximum distinct characters. Note : Our output can have same character.



Examples:

```
Input : "AABBBCBB"
Output : 5

Input : "AABBBCBBAC"
Output : 3
Explanation : Sub-string -> "BAC"

Input : "GEEKSGEEKSFOR"
Output : 8
Explanation : Sub-string -> "GEEKSFOR"
```

Method 1 (Brute Force)

We can consider all sub-strings one by one and check for each sub-string both conditions together

1. sub-string's distinct characters is equal to maximum distinct characters
2. sub-sting's length should be minimum .

Time Complexity : O(n^3)

```
 /* C++ program to find the length of the smallest
substring consisting of maximum distinct characters */
#include <bits/stdc++.h>
using namespace std;

#define NO_OF_CHARS 256

// Find maximum distinct characters in any string
int max_distinct_char(string str, int n){

    // Initialize all character's count with 0
    int count[NO_OF_CHARS] = {0};

    // Increase the count in array if a character
    // is found
    for (int i = 0; i < n;  i++)
        count[str[i]]++;

    int max_distinct = 0;
    for (int i = 0; i < NO_OF_CHARS;  i++)
        if (count[i] != 0)
            max_distinct++;

    return max_distinct;
}

int smallesteSubstr_maxDistictChar(string str){

    int n = str.size();     // size of given string

    // Find maximum distinct characters in any string
    int max_distinct = max_distinct_char(str, n);
    int minl = n;   // result

    // Brute force approch to find all substrings
    for (int i=0 ;i<n ;i++){
        for (int j=0; j<n; j++){
            string subs =  str.substr(i,j);
            int subs_lenght = subs.size();
            int sub_distinct_char = max_distinct_char(subs, subs_lenght);

            // We have to check here both conditions together
            // 1. substring's distinct characters is equal
            //    to maximum distinct characters
```

```
            // 2. substing's length should be minimum
            if (subs_lenght < minl && max_distinct == sub_distinct_char){
                minl = subs_lenght;
            }
        }
    }
    return minl;
}

/* Driver program to test above function */
int main()
{
    // Input String
    string str = "AABBBCBB";

    int len =  smallesteSubstr_maxDistictChar(str);
    cout << " The length of the smallest substring"
            " consisting of maximum distinct "
            "characters : " << len;
    return 0;
}
```

Output:

```
The length of the smallest substring consisting
of maximum distinct characters : 5
```

Method 2 (Efficient)

1. Count all distinct characters in given string.
2. Maintain a window of characters. Whenever the window contains all characters of given string, we shrink the window from left side to remove extra characters and then compare its length with smallest window fount so far.

Please refer Smallest window that contains all characters of string itself for implementation and more details.

Asked In : DailyHunt

Source

https://www.geeksforgeeks.org/length-smallest-sub-string-consisting-maximum-distinct-characters/

Chapter 173

Linked List Pair Sum

Linked List Pair Sum - GeeksforGeeks

Given a linked list, and a number, check if their exist two numbers whose sum is equal to given number. If there exist two numbers, print them. If there are multiple answer, print any of them.

Examples:

```
Input : 1 -> 2 -> 3 -> 4 -> 5 -> NULL
        sum = 3
Output : Pair is (1, 2)

Input : 10 -> 12 -> 31 -> 42 -> 53 -> NULL
        sum = 15
Output : NO PAIR EXIST
```

Method(Brute force)
Iteratively check if their exist any pair or not

C++

```
 // CPP code to find the pair with given sum
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* Given a reference (pointer to pointer)
```

```
    to the head of a list and an int,
    push a new node on the front
    of the list. */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
          (struct Node*)malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Takes head pointer of the linked list and sum*/
int check_pair_sum(struct Node* head, int sum)
{
    struct Node* p = head, *q;
    while (p != NULL) {

        q = p->next;
        while (q != NULL) {

           // check if both sum is equal to
           // given sum
           if ((p->data) + (q->data) == sum) {
               cout << p->data << " " << q->data;
               return true;
           }
           q = q->next;
        }

        p = p->next;
    }

    return 0;
}

/* Driver program to test above function */
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;
```

```
    /* Use push() to construct linked list*/
    push(&head, 1);
    push(&head, 4);
    push(&head, 1);
    push(&head, 12);
    push(&head, 1);
    push(&head, 18);
    push(&head, 47);
    push(&head, 16);
    push(&head, 12);
    push(&head, 14);

    /* function to print the result*/
    bool res = check_pair_sum(head, 26);
    if (res == false)
        cout << "NO PAIR EXIST";

    return 0;
}
```

Output:

```
14 12
```

Time complexity:O(n*n)

Method 2 (using hashing)
1. Take a hashtable and mark all element with zero
2. Iteratively mark all the element as 1 in hashtable which are present in linked list
3. Iteratively find sum-current element of linked list is present in hashtable or not

C++

```
 // CPP program to for finding the pair with given sum
#include <bits/stdc++.h>
#define MAX 100000
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* Given a reference (pointer to pointer) to the head
    of a list and an int, push a new node on the front
```

```
    of the list. */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
            (struct Node*)malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Takes head pointer of the linked list and sum*/
bool check_pair_sum(struct Node* head, int sum)
{
    unordered_set<int> s;

    struct Node* p = head;
    while (p != NULL) {
        int curr = p->data;
        if (s.find(sum - curr) != s.end())
        {
           cout << curr << " " << sum - curr;
           return true;
        }
        s.insert(p->data);
        p = p->next;
    }

    return false;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    /* Use push() to construct linked list */
    push(&head, 1);
    push(&head, 4);
    push(&head, 1);
    push(&head, 12);
```

```
    push(&head, 1);
    push(&head, 18);
    push(&head, 47);
    push(&head, 16);
    push(&head, 12);
    push(&head, 14);

    /* function to print the result*/
    bool res = check_pair_sum(head, 26);
    if (res == false)
        cout << "NO PAIR EXIST";

    return 0;
}
```

Output:

```
14 12
```

Time complexity : O(n)
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/linked-list-pair-sum/

Chapter 174

Load Factor and Rehashing

Load Factor and Rehashing - GeeksforGeeks

Prerequisites: Hashing Introduction and Collision handling by separate chaining

For **insertion** of a key(K) – value(V) pair into a hash map, 2 steps are required:

1. K is converted into a small integer (called its hash code) using a hash function.
2. The hash code is used to find an index (hashCode % arrSize) and the entire linked list at that index(Separate chaining) is first searched for the presence of the K already.
3. If found, it's value is updated and if not, the K-V pair is stored as a new node in the list.

- For the **first step**, time taken depends on the K and the hash function.
  For example, if the key is a string "abcd", then it's hash function may depend on the length of the string. But for very large values of n, the number of entries into the map, length of the keys is almost negligible in comparison to n so hash computation can be considered to take place in constant time, i.e, **O(1)**.
- For the **second step**, traversal of the list of K-V pairs present at that index needs to be done. For this, the worst case may be that all the n entries are at the same index. So, time complexity would be **O(n)**. But, enough research has been done to make hash functions uniformly distribute the keys in the array so this almost never happens.
- So, **on an average**, if there are n entries and b is the size of the array there would be n/b entries on each index. This value n/b is called the **load factor** that represents the load that is there on our map.
- This Load Factor needs to be kept low, so that number of entries at one index is less and so is the complexity almost constant, i.e., O(1).

As the name suggests, **rehashing means hashing again**. Basically, when the load factor increases to more than its pre-defined value (default value of load factor is 0.75), the complexity increases. So to overcome this, the size of the array is increased (doubled) and all the values are hashed again and stored in the new double sized array to maintain a low load factor and low complexity.

Why rehashing?

Rehashing is done because whenever key value pairs are inserted into the map, the load factor increases, which implies that the time complexity also increases as explained above. This might not give the required time complexity of O(1).

Hence, rehash must be done, increasing the size of the bucketArray so as to reduce the load factor and the time complexity.

How Rehashing is done?

Rehashing can be done as follows:

- For each addition of a new entry to the map, check the load factor.
- If it's greater than its pre-defined value (or default value of 0.75 if not given), then Rehash.
- For Rehash, make a new array of double the previous size and make it the new bucketarray.
- Then traverse to each element in the old bucketArray and call the insert() for each so as to insert it into the new larger bucket array.

```
 // Java program to implement Rehashing

import java.util.ArrayList;

class Map<K, V> {

    class MapNode<K, V> {

        K key;
        V value;
        MapNode<K, V> next;

        public MapNode(K key, V value)
        {
            this.key = key;
            this.value = value;
            next = null;
        }
    }

    // The bucket array where
    // the nodes containing K-V pairs are stored
    ArrayList<MapNode<K, V> > buckets;

    // No. of pairs stored - n
    int size;
```

```
// Size of the bucketArray - b
int numBuckets;

// Default loadFactor
final double DEFAULT_LOAD_FACTOR = 0.75;

public Map()
{
    numBuckets = 5;

    buckets = new ArrayList<>(numBuckets);

    for (int i = 0; i < numBuckets; i++) {
        // Initialising to null
        buckets.add(null);
    }
    System.out.println("HashMap created");
    System.out.println("Number of pairs in the Map: " + size);
    System.out.println("Size of Map: " + numBuckets);
    System.out.println("Default Load Factor : " + DEFAULT_LOAD_FACTOR + "\n");
}

private int getBucketInd(K key)
{

    // Using the inbuilt function from the object class
    int hashCode = key.hashCode();

    // array index = hashCode%numBuckets
    return (hashCode % numBuckets);
}

public void insert(K key, V value)
{
    // Getting the index at which it needs to be inserted
    int bucketInd = getBucketInd(key);

    // The first node at that index
    MapNode<K, V> head = buckets.get(bucketInd);

    // First, loop through all the nodes present at that index
    // to check if the key already exists
    while (head != null) {

        // If already present the value is updated
        if (head.key.equals(key)) {
            head.value = value;
            return;
```

```
            }
            head = head.next;
        }

        // new node with the K and V
        MapNode<K, V> newElementNode = new MapNode<K, V>(key, value);

        // The head node at the index
        head = buckets.get(bucketInd);

        // the new node is inserted
        // by making it the head
        // and it's next is the previous head
        newElementNode.next = head;

        buckets.set(bucketInd, newElementNode);

        System.out.println("Pair(" + key + ", " + value + ") inserted successfully.\n");

        // Incrementing size
        // as new K-V pair is added to the map
        size++;

        // Load factor calculated
        double loadFactor = (1.0 * size) / numBuckets;

        System.out.println("Current Load factor = " + loadFactor);

        // If the load factor is > 0.75, rehashing is done
        if (loadFactor > DEFAULT_LOAD_FACTOR) {
            System.out.println(loadFactor + " is greater than " + DEFAULT_LOAD_FACTOR);
            System.out.println("Therefore Rehashing will be done.\n");

            // Rehash
            rehash();

            System.out.println("New Size of Map: " + numBuckets + "\n");
        }

        System.out.println("Number of pairs in the Map: " + size);
        System.out.println("Size of Map: " + numBuckets + "\n");
    }

    private void rehash()
    {

        System.out.println("\n***Rehashing Started***\n");
```

```
    // The present bucket list is made temp
    ArrayList<MapNode<K, V> > temp = buckets;

    // New bucketList of double the old size is created
    buckets = new ArrayList<MapNode<K, V> >(2 * numBuckets);

    for (int i = 0; i < 2 * numBuckets; i++) {
        // Initialised to null
        buckets.add(null);
    }
    // Now size is made zero
    // and we loop through all the nodes in the original bucket list(temp)
    // and insert it into the new list
    size = 0;
    numBuckets *= 2;

    for (int i = 0; i < temp.size(); i++) {

        // head of the chain at that index
        MapNode<K, V> head = temp.get(i);

        while (head != null) {
            K key = head.key;
            V val = head.value;

            // calling the insert function for each node in temp
            // as the new list is now the bucketArray
            insert(key, val);
            head = head.next;
        }
    }

    System.out.println("\n***Rehashing Ended***\n");
}

public void printMap()
{

    // The present bucket list is made temp
    ArrayList<MapNode<K, V> > temp = buckets;

    System.out.println("Current HashMap:");
    // loop through all the nodes and print them
    for (int i = 0; i < temp.size(); i++) {

        // head of the chain at that index
        MapNode<K, V> head = temp.get(i);
```

```
            while (head != null) {
                System.out.println("key = " + head.key + ", val = " + head.value);

                head = head.next;
            }
        }
        System.out.println();
    }
}

public class GFG {

    public static void main(String[] args)
    {

        // Creating the Map
        Map<Integer, String> map = new Map<Integer, String>();

        // Inserting elements
        map.insert(1, "Geeks");
        map.printMap();

        map.insert(2, "forGeeks");
        map.printMap();

        map.insert(3, "A");
        map.printMap();

        map.insert(4, "Computer");
        map.printMap();

        map.insert(5, "Portal");
        map.printMap();
    }
}
```

Output:

```
HashMap created
Number of pairs in the Map: 0
Size of Map: 5
Default Load Factor : 0.75

Pair(1, Geeks) inserted successfully.

Current Load factor = 0.2
Number of pairs in the Map: 1
```

```
Size of Map: 5

Current HashMap:
key = 1, val = Geeks

Pair(2, forGeeks) inserted successfully.

Current Load factor = 0.4
Number of pairs in the Map: 2
Size of Map: 5

Current HashMap:
key = 1, val = Geeks
key = 2, val = forGeeks

Pair(3, A) inserted successfully.

Current Load factor = 0.6
Number of pairs in the Map: 3
Size of Map: 5

Current HashMap:
key = 1, val = Geeks
key = 2, val = forGeeks
key = 3, val = A

Pair(4, Computer) inserted successfully.

Current Load factor = 0.8
0.8 is greater than 0.75
Therefore Rehashing will be done.


***Rehashing Started***

Pair(1, Geeks) inserted successfully.

Current Load factor = 0.1
Number of pairs in the Map: 1
Size of Map: 10

Pair(2, forGeeks) inserted successfully.

Current Load factor = 0.2
Number of pairs in the Map: 2
Size of Map: 10

Pair(3, A) inserted successfully.
```

```
Current Load factor = 0.3
Number of pairs in the Map: 3
Size of Map: 10

Pair(4, Computer) inserted successfully.

Current Load factor = 0.4
Number of pairs in the Map: 4
Size of Map: 10


***Rehashing Ended***

New Size of Map: 10

Number of pairs in the Map: 4
Size of Map: 10

Current HashMap:
key = 1, val = Geeks
key = 2, val = forGeeks
key = 3, val = A
key = 4, val = Computer

Pair(5, Portal) inserted successfully.

Current Load factor = 0.5
Number of pairs in the Map: 5
Size of Map: 10

Current HashMap:
key = 1, val = Geeks
key = 2, val = forGeeks
key = 3, val = A
key = 4, val = Computer
key = 5, val = Portal
```

Source

https://www.geeksforgeeks.org/load-factor-and-rehashing/

Chapter 175

Longest Consecutive Subsequence

Longest Consecutive Subsequence - GeeksforGeeks

Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Examples

```
Input: arr[] = {1, 9, 3, 10, 4, 20, 2};
Output: 4
The subsequence 1, 3, 4, 2 is the longest subsequence
of consecutive elements

Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42}
Output: 5
The subsequence 36, 35, 33, 34, 32 is the longest subsequence
of consecutive elements.
```

One Solution is to first sort the array and find the longest subarray with consecutive elements. Time complexity of this solution is O(nLogn). Thanks to Hao.W for suggesting this solution.

We can solve this problem in O(n) time using an **Efficient Solution**. The idea is to use Hashing. We first insert all elements in a Hash. Then check all the possible starts of consecutive subsequences. Below is complete algorithm.

```
1) Create an empty hash.
2) Insert all array elements to hash.
3) Do following for every element arr[i]
```

```
....a) Check if this element is the starting point of a
       subsequence.  To check this, we simply look for
       arr[i] - 1 in hash, if not found, then this is
       the first element a subsequence.

       If this element is a first element, then count
       number of elements in the consecutive starting
       with this element.

       If count is more than current res, then update
       res.
```

Below is C++ implementation of above algorithm.

C/C++

```
 // C++ program to find longest contiguous subsequence
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest contiguous subsequence
int findLongestConseqSubseq(int arr[], int n)
{
    unordered_set<int> S;
    int ans = 0;

    // Hash all the array elements
    for (int i = 0; i < n; i++)
        S.insert(arr[i]);

    // check each possible sequence from the start
    // then update optimal length
    for (int i=0; i<n; i++)
    {
        // if current element is the starting
        // element of a sequence
        if (S.find(arr[i]-1) == S.end())
        {
            // Then check for next elements in the
            // sequence
            int j = arr[i];
            while (S.find(j) != S.end())
                j++;

            // update  optimal length if this length
            // is more
            ans = max(ans, j - arr[i]);
        }
```

```
    }
    return ans;
}

// Driver program
int main()
{
    int arr[] =  {1, 9, 3, 10, 4, 20, 2};
    int n = sizeof arr/ sizeof arr[0];
    cout << "Length of the Longest contiguous subsequence is "
         << findLongestConseqSubseq(arr, n);
    return 0;
}
```

Java

```
 // Java program to find longest consecutive subsequence
import java.io.*;
import java.util.*;

class ArrayElements
{
    // Returns length of the longest consecutive subsequence
    static int findLongestConseqSubseq(int arr[],int n)
    {
        HashSet<Integer> S = new HashSet<Integer>();
        int ans = 0;

        // Hash all the array elements
        for (int i=0; i<n; ++i)
            S.add(arr[i]);

        // check each possible sequence from the start
        // then update optimal length
        for (int i=0; i<n; ++i)
        {
            // if current element is the starting
            // element of a sequence
            if (!S.contains(arr[i]-1))
            {
                // Then check for next elements in the
                // sequence
                int j = arr[i];
                while (S.contains(j))
                    j++;

                // update  optimal length if this length
                // is more
```

```
                if (ans<j-arr[i])
                    ans = j-arr[i];
            }
        }
        return ans;
    }

    // Testing program
    public static void main(String args[])
    {
        int arr[] =  {1, 9, 3, 10, 4, 20, 2};
        int n = arr.length;
        System.out.println("Length of the Longest consecutive subsequence is " +
                           findLongestConseqSubseq(arr,n));
    }
}
// This code is contributed by Aakash Hasija
```

Python

```
 # Python program to find longest contiguous subsequence

from sets import Set
def findLongestConseqSubseq(arr, n):

    s = Set()
    ans=0

    # Hash all the array elements
    for ele in arr:
        s.add(ele)

    # check each possible sequence from the start
    # then update optimal length
    for i in range(n):

         # if current element is the starting
        # element of a sequence
        if (arr[i]-1) not in s:

            # Then check for next elements in the
            # sequence
            j=arr[i]
            while(j in s):
                j+=1

            # update  optimal length if this length
            # is more
```

```
            ans=max(ans, j-arr[i])
    return ans

# Driver function
if __name__=='__main__':
    n = 7
    arr = [1, 9, 3, 10, 4, 20, 2]
    print "Length of the Longest contiguous subsequence is ",
    print  findLongestConseqSubseq(arr, n)

# Contributed by: Harshit Sidhwa
```

Output:

```
Length of the Longest contiguous subsequence is 4
```

Time Complexity: At first look, time complexity looks more than O(n). If we take a closer look, we can notice that it is O(n) under the assumption that hash insert and search take O(1) time. The function S.find() inside the while loop is called at most twice for every element. For example, consider the case when all array elements are consecutive. In this case, the outer find is called for every element, but we go inside the if condition only for the smallest element. Once we are inside the if condition, we call find() one more time for every other element.

Thanks to Gaurav Ahirwar for above solution.

Source

https://www.geeksforgeeks.org/longest-consecutive-subsequence/

Chapter 176

Longest Increasing consecutive subsequence

Longest Increasing consecutive subsequence - GeeksforGeeks

Given N elements, write a program that prints the length of the longest increasing subsequence whose adjacent element difference is one.

Examples:

> Input : a[] = {3, 10, 3, 11, 4, 5, 6, 7, 8, 12}
> Output : 6
> Explanation: 3, 4, 5, 6, 7, 8 is the longest increasing subsequence whose adjacent element differs by one.
>
> Input : a[] = {6, 7, 8, 3, 4, 5, 9, 10}
> Output : 5
> Explanation: 6, 7, 8, 9, 10 is the longest increasing subsequence

Naive Approach: A normal approach will be to iterate for every element and find out the longest increasing subsequence. For any particular element, find the length of the subsequence starting from that element. Print the longest length of the subsequence thus formed. The time complexity of this approach will be $O(n^2)$.

Dynamic Programming Approach: Let DP[i] store the length of the longest subsequence which ends with A[i]. For every A[i], if A[i]-1 is present in the array before i-th index, then A[i] will add to the increasing subsequence which has A[i]-1. Hence **DP[i] = DP[index(A[i]-1)] + 1**. If A[i]-1 is not present in the array before i-th index, then DP[i]=1 since the A[i] element forms a subsequence which starts with A[i]. Hence the relation for DP[i] is:

> If A[i]-1 is present before i-th index:
>
> DP[i] = DP[index(A[i]-1)] + 1

else:

DP[i] = 1

Given below is the illustration of the above approach:

```
 // CPP program to find length of the
// longest increasing subsequence
// whose adjacent element differ by 1
#include <bits/stdc++.h>
using namespace std;

// function that returns the length of the
// longest increasing subsequence
// whose adjacent element differ by 1
int longestSubsequence(int a[], int n)
{
    // stores the index of elements
    unordered_map<int, int> mp;

    // stores the length of the longest
    // subsequence that ends with a[i]
    int dp[n];
    memset(dp, 0, sizeof(dp));

    int maximum = INT_MIN;

    // iterate for all element
    for (int i = 0; i < n; i++) {

        // if a[i]-1 is present before i-th index
        if (mp.find(a[i] - 1) != mp.end()) {

            // last index of a[i]-1
            int lastIndex = mp[a[i] - 1] - 1;

            // relation
            dp[i] = 1 + dp[lastIndex];
        }
        else
            dp[i] = 1;

        // stores the index as 1-index as we need to
        // check for occurrence, hence 0-th index
        // will not be possible to check
        mp[a[i]] = i + 1;

        // stores the longest length
```

```
        maximum = max(maximum, dp[i]);
    }

    return maximum;
}

// Driver Code
int main()
{
    int a[] = { 3, 10, 3, 11, 4, 5, 6, 7, 8, 12 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << longestSubsequence(a, n);
    return 0;
}
```

Output:

```
6
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/longest-increasing-consecutive-subsequence/

Chapter 177

Longest string in non-decreasing order of ASCII code and in arithmetic progression

Longest string in non-decreasing order of ASCII code and in arithmetic progression - GeeksforGeeks

Given a non-empty string S of uppercase alphabets of length L and the task is to find the longest string from the given string with characters arranged in descending order of their ASCII code and in arithmetic progression such that the common difference should be as low as possible and the characters of the string to be of higher ASCII value.
Note : The string contains minimum three different characters.

Examples:

> **Input** : S = "ABCPQR"
> **Output** : "RQP"
> Two strings of maximum length are possible – "CBA" and "RPQ". But since the string should be of higher ASCII value hence, the output is "RPQ".
>
> **Input** : S = "ADGJPRT"
> **Output** : "JGDA"

Approach : The maximum possible common difference for minimum 3 characters to be in arithmetic progression is 12. Hence, precompute all characters that are present in the string using a hashmap and then iterate from the character having maximum ASCII value i.e. 'Z' to the character having minimum ASCII value i.e. 'A'. If the current character exists in the given string, consider it as the starting character of the arithmetic progression sequence and iterate again over all possible common differences i.e. from 1 to 12. Check for every current common difference that if the character exists in the given string, increment the current length of the longest required string. Now, there exist two cases when maximum length *ans* minimum common difference needs to be updated.

1. When the current length is more than the maximum length.
2. When the current length is equal to the maximum length and current common difference is less than the minimum common difference, then common difference needs to be updated.

Also, at every updation of these two parameters, starting character of the string or arithmetic progression sequence must also be updated.

Below is the implementation of above approach:

C++

```
 // C++ Program to find the longest string
// with characters arranged in non-decreasing
// order of ASCII and in arithmetic progression
#include <bits/stdc++.h>
using namespace std;

// Function to find the longest String
string findLongestString(string S)
{
    // Stores the maximum length of required string
    int maxLen = 0;

    // Stores the optimal starting character of
    // required string or arithmetic progression sequence
    int bestStartChar;

    // Stores the optimal i.e. minimum common difference
    // of required string
    int minCommonDifference = INT_MAX;

    unordered_map<char, bool> mp;
    for (int i = 0; i < S.size(); i++)
        mp[S[i]] = true;

    // Iterate over the loop in non decreasing order
    for (int startChar = 'Z'; startChar > 'A'; startChar--) {

        // Process further only if current character
        // exists in the given string
        if (mp[startChar]) {

            // Iterate over all possible common differences
            // of AP sequence and update maxLen accordingly
            for (int currDiff = 1; currDiff <= 12; currDiff++) {
                int currLen = 1;
```

```
                // Iterate over the characters at any interval
                // of current common difference
                for (int ch = startChar - currDiff; ch >= 'A';
                     ch -= currDiff) {
                    if (mp[ch])
                        currLen++;
                    else
                        break;
                }

                // Update maxLen and other parameters if the currLen
                // is greater than maxLen or if the current
                // difference is smaller than minCommonDifference
                if (currLen > maxLen || (currLen == maxLen
                                         && currDiff < minCommonDifference)) {
                    minCommonDifference = currDiff;
                    maxLen = currLen;
                    bestStartChar = startChar;
                }
            }
        }
    }
    string longestString = "";

    // Store the string in decreasing order of
    // arithmetic progression
    for (int i = bestStartChar;
         i >= (bestStartChar - (maxLen - 1) * minCommonDifference);
         i -= minCommonDifference)
        longestString += char(i);

    return longestString;
}

// Driver Code
int main()
{
    string S = "ADGJPRT";
    cout << findLongestString(S) << endl;
    return 0;
}
```

Java

```
 // Java Program to find the longest string
// with characters arranged in non-decreasing
// order of ASCII and in arithmetic progression
import java.util.*;
```

```
import java.lang.*;

public class GFG {
    // Function to find the longest String
    static String findLongestString(String S)
    {
        // Stores the maximum length of required string
        int maxLen = 0;

        // Stores the optimal starting character of
        // required string or arithmetic progression sequence
        int bestStartChar = 0;

        // Stores the optimal i.e. minimum common difference
        // of required string
        int minCommonDifference = Integer.MAX_VALUE;

        HashMap <Character, Boolean> hm = new HashMap
                                <Character, Boolean>();
        for (int i = 0; i < S.length(); i++)
            hm.put(S.charAt(i), true);

        // Iterate over the loop in non decreasing order
        for (int startChar = 'Z'; startChar > 'A'; startChar--) {

            // Process further only if current character
            // exists in the given string
            if (hm.containsKey((char)startChar)) {

                // Iterate over all possible common differences
                // of AP sequence and update maxLen accordingly
                for (int currDiff = 1; currDiff <= 12; currDiff++) {
                    int currLen = 1;

                    // Iterate over the characters at any interval
                    // of current common difference
                    for (int ch = startChar - currDiff; ch >= 'A';
                        ch -= currDiff) {
                        if (hm.containsKey((char)ch))
                            currLen++;
                        else
                            break;
                    }

                    // Update maxLen and other parameters if the currLen
                    // is greater than maxLen or if the current
                    // difference is smaller than minCommonDifference
                    if (currLen > maxLen || (currLen == maxLen
```

```
                                && currDiff < minCommonDifference)) {
                        minCommonDifference = currDiff;
                        maxLen = currLen;
                        bestStartChar = startChar;
                    }
                }
            }
        }
        String longestString = "";

        // Store the string in decreasing order of
        // arithmetic progression
        char ch;
        for (int i = bestStartChar;
         i >= (bestStartChar - (maxLen - 1) * minCommonDifference);
         i -= minCommonDifference)
        {
            ch = (char)i;
            longestString += ch;
        }
        return longestString;
    }

    // Driver Code
    public static void main(String args[])
    {
        String S = "ABCPQR";
        System.out.println(findLongestString(S));
    }
}
// This code is contributed by Nishant Tanwar
```

Output:

```
JGDA
```

Time Complexity : O(|S| + 26*12*26), where |S| is the size of the string.

Improved By : Nishant Tanwar

Source

https://www.geeksforgeeks.org/longest-string-in-non-decreasing-order-of-ascii-code-and-in-arithmetic-progression/

Chapter 178

Longest sub-array having sum k

Longest sub-array having sum k - GeeksforGeeks

Given an array **arr**[] of size **n** containing integers. The problem is to find the length of the longest sub-array having sum equal to the given value **k**.

Examples:

```
Input : arr[] = { 10, 5, 2, 7, 1, 9 },
            k = 15
Output : 4
The sub-array is {5, 2, 7, 1}.

Input : arr[] = {-5, 8, -14, 2, 4, 12},
            k = -5
Output : 5
```

Naive Approach: Consider the sum of all the sub-arrays and return the length of the longest sub-array having sum 'k'. Time Complexity is of O(n^2).

Efficient Approach: Following are the steps:

1. Initialize **sum** = 0 and **maxLen** = 0.
2. Create a hash table having **(sum, index)** tuples.
3. For i = 0 to n-1, perform the following steps:
   (a) Accumulate arr[i] to **sum**.
   (b) If sum == k, update **maxLen** = i+1.
   (c) Check whether **sum** is present in the hash table or not. If not present, then add it to the hash table as **(sum, i)** pair.
   (d) Check if **(sum-k)** is present in the hash table or not. If present, then obtain index of **(sum-k)** from the hash table as **index**. Now check if maxLen < (i-index), then update **maxLen** = (i-index).

4. Return **maxLen**.

```
 // C++ implementation to find the length
// of longest subarray having sum k
#include <bits/stdc++.h>
using namespace std;

// function to find the length of longest
// subarray having sum k
int lenOfLongSubarr(int arr[],
                    int n,
                    int k)
{

    // unordered_map 'um' implemented
    // as hash table
    unordered_map<int, int> um;
    int sum = 0, maxLen = 0;

    // traverse the given array
    for (int i = 0; i < n; i++) {

        // accumulate sum
        sum += arr[i];

        // when subarray starts from index '0'
        if (sum == k)
            maxLen = i + 1;

        // make an entry for 'sum' if it is
        // not present in 'um'
        if (um.find(sum) == um.end())
            um[sum] = i;

        // check if 'sum-k' is present in 'um'
        // or not
        if (um.find(sum - k) != um.end()) {

            // update maxLength
            if (maxLen < (i - um[sum - k]))
                maxLen = i - um[sum - k];
        }
    }

    // required maximum length
    return maxLen;
}
```

```
// Driver Code
int main()
{
    int arr[] = {10, 5, 2, 7, 1, 9};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 15;
    cout << "Length = "
         << lenOfLongSubarr(arr, n, k);
    return 0;
}
```

Output:

```
Length = 4
```

Time Complexity: O(n).
Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/longest-sub-array-sum-k/

Chapter 179

Longest subarray having count of 1s one more than count of 0s

Longest subarray having count of 1s one more than count of 0s - GeeksforGeeks

Given an array of size **n** containing 0's and 1's only. The problem is to find the length of the longest subarray having count of 1's one more than count of 0's.

Examples:

```
Input : arr = {0, 1, 1, 0, 0, 1}
Output : 5
From index 1 to 5.

Input : arr[] = {1, 0, 0, 1, 0}
Output : 1
```

Approach: Following are the steps:

1. Consider all the 0's in the array as '-1'.
2. Initialize **sum** = 0 and **maxLen** = 0.
3. Create a hash table having **(sum, index)** tuples.
4. For i = 0 to n-1, perform the following steps:
   (a) If arr[i] is '0' accumulate '-1' to **sum** else accumulate '1' to **sum**.
   (b) If sum == 1, update **maxLen** = i+1.
   (c) Else check whether **sum** is present in the hash table or not. If not present, then add it to the hash table as **(sum, i)** pair.
   (d) Check if **(sum-1)** is present in the hash table or not. if present, then obtain index of **(sum-1)** from the hash table as **index**. Now check if maxLen is less than (i-index), then update **maxLen** = (i-index).
5. Return maxLen.

```
 // C++ implementation to find the length of
// longest subarray having count of 1's one
// more than count of 0's
#include <bits/stdc++.h>
using namespace std;

// function to find the length of longest
// subarray having count of 1's one more
// than count of 0's
int lenOfLongSubarr(int arr[], int n)
{
    // unordered_map 'um' implemented as
    // hash table
    unordered_map<int, int> um;
    int sum = 0, maxLen = 0;

    // traverse the given array
    for (int i = 0; i < n; i++) {

        // consider '0' as '-1'
        sum += arr[i] == 0 ? -1 : 1;

        // when subarray starts form index '0'
        if (sum == 1)
            maxLen = i + 1;

        // make an entry for 'sum' if it is
        // not present in 'um'
        else if (um.find(sum) == um.end())
            um[sum] = i;

        // check if 'sum-1' is present in 'um'
        // or not
        if (um.find(sum - 1) != um.end()) {

            // update maxLength
            if (maxLen < (i - um[sum - 1]))
                maxLen = i - um[sum - 1];
        }
    }

    // required maximum length
    return maxLen;
}

// Driver program to test above
int main()
{
```

```
    int arr[] = { 0, 1, 1, 0, 0, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Length = "
         << lenOfLongSubarr(arr, n);
    return 0;
}
```

Output:

```
Length = 5
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Source

https://www.geeksforgeeks.org/longest-subarray-count-1s-one-count-0s/

Chapter 180

Longest subarray having maximum sum

Longest subarray having maximum sum - GeeksforGeeks

Given an array **arr**[] containing **n** integers. The problem is to find the length of the subarray having maximum sum. If there exists two or more subarrays with maximum sum then print the length of the longest subarray.

Examples:

```
Input : arr[] = {5, -2, -1, 3, -4}
Output : 4
There are two subarrays with maximum sum:
First is {5}
Second is {5, -2, -1, 3}
Therefore longest one is of length 4.

Input : arr[] = {-2, -3, 4, -1, -2, 1, 5, -3}
Output : 5
The subarray is {4, -1, -2, 1, 5}
```

Approach: Following are the steps:

1. Find the maximum sum contiguous subarray. Let this sum be **maxSum**.
2. Find the length of the longest subarray having sum equal to **maxSum**. Refer this post.

```
 // C++ implementation to find the length of the longest
// subarray having maximum sum
#include <bits/stdc++.h>
```

```
using namespace std;

// function to find the maximum sum that
// exists in a subarray
int maxSubArraySum(int arr[], int size)
{
    int max_so_far = arr[0];
    int curr_max = arr[0];

    for (int i = 1; i < size; i++) {
        curr_max = max(arr[i], curr_max + arr[i]);
        max_so_far = max(max_so_far, curr_max);
    }
    return max_so_far;
}

// function to find the length of longest
// subarray having sum k
int lenOfLongSubarrWithGivenSum(int arr[], int n, int k)
{
    // unordered_map 'um' implemented
    // as hash table
    unordered_map<int, int> um;
    int sum = 0, maxLen = 0;

    // traverse the given array
    for (int i = 0; i < n; i++) {

        // accumulate sum
        sum += arr[i];

        // when subarray starts from index '0'
        if (sum == k)
            maxLen = i + 1;

        // make an entry for 'sum' if it is
        // not present in 'um'
        if (um.find(sum) == um.end())
            um[sum] = i;

        // check if 'sum-k' is present in 'um'
        // or not
        if (um.find(sum - k) != um.end()) {

            // update maxLength
            if (maxLen < (i - um[sum - k]))
                maxLen = i - um[sum - k];
        }
```

```
    }

    // required maximum length
    return maxLen;
}

// function to find the length of the longest
// subarray having maximum sum
int lenLongSubarrWithMaxSum(int arr[], int n)
{
    int maxSum = maxSubArraySum(arr, n);
    return lenOfLongSubarrWithGivenSum(arr, n, maxSum);
}

// Driver program to test above
int main()
{
    int arr[] = { 5, -2, -1, 3, -4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Length of longest subarray having maximum sum = "
         << lenLongSubarrWithMaxSum(arr, n);
    return 0;
}
```

Output:

```
Length of longest subarray having maximum sum = 4
```

Time Complexity: O(n).
Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/longest-subarray-having-maximum-sum/

Chapter 181

Longest subarray not having more than K distinct elements

Longest subarray not having more than K distinct elements - GeeksforGeeks

Given N elements and a number K, find the longest subarray which has not more than K distinct elements.(It can have less than K)

Examples:

```
Input : arr[] = {1, 2, 3, 4, 5}
            k = 6
Output : 1 2 3 4 5
Explanation: The whole array has only 5
distinct elements which is less than k,
so we print the array itself.

Input: arr[] = {6, 5, 1, 2, 3, 2, 1, 4, 5}
           k = 3
Output: 1 2 3 2 1,
The output is the longest subarray with 3
distinct elements.
```

A **naive approach** will be to be traverse in the array and use hashing for every sub-arrays, and check for the longest sub-array possible with no more than K distinct elements.

An **efficient approach** is to use the concept of *two pointers* where we maintain a hash to count for occurrences of elements. We start from the beginning and keep a count of distinct elements till the number exceeds k. Once it exceeds K, we start decreasing the count of the elements in the hash from where the sub-array started and reduce our length as the sub-arrays gets decreased so the pointer moves to the right. We keep removing elements till we again get k distinct elements. We continue this process till we again have more than k

distinct elements and keep the left pointer constant till then. We update our start and end according to that if the new sub-array length is more than the previous one.

```
 // CPP program to find longest subarray with
// k or less distinct elements.
#include <bits/stdc++.h>
using namespace std;

// function to print the longest sub-array
void longest(int a[], int n, int k)
{
    unordered_map<int, int> freq;

    int start = 0, end = 0, now = 0, l = 0;
    for (int i = 0; i < n; i++) {

        // mark the element visited
        freq[a[i]]++;

        // if its visited first time, then increase
        // the counter of distinct elements by 1
        if (freq[a[i]] == 1)
            now++;

        // When the counter of distinct elements
        // increases from k, then reduce it to k
        while (now > k) {

            // from the left, reduce the number of
            // time of visit
            freq[a[l]]--;

            // if the reduced visited time element
            // is not present in further segment
            // then decrease the count of distinct
            // elements
            if (freq[a[l]] == 0)
                now--;

            // increase the subsegment mark
            l++;
        }

        // check length of longest sub-segment
        // when greater then previous best
        // then change it
        if (i - l + 1 >= end - start + 1)
            end = i, start = l;
```

```
    }

    // print the longest sub-segment
    for (int i = start; i <= end; i++)
        cout << a[i] << " ";
}

// driver program to test the above function
int main()
{
    int a[] = { 6, 5, 1, 2, 3, 2, 1, 4, 5 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 3;
    longest(a, n, k);
    return 0;
}
```

Output:

```
1 2 3 2 1
```

Time Complexity: O(n)

Improved By : suraznegi

Source

https://www.geeksforgeeks.org/longest-subarray-not-k-distinct-elements/

Chapter 182

Longest subarray with sum divisible by k

Longest subarray with sum divisible by k - GeeksforGeeks

Given an **arr[]** containing **n** integers and a positive integer **k**. The problem is to find the length of the longest subarray with sum of the elements divisible by the given value **k**.

Examples:

```
Input : arr[] = {2, 7, 6, 1, 4, 5}, k = 3
Output : 4
The subarray is {7, 6, 1, 4} with sum 18,
which is divisible by 3.

Input : arr[] = {-2, 2, -5, 12, -11, -1, 7}
Output : 5
```

Method 1 (Naive Approach): Consider all the subarrays and return the length of the subarray with sum divisible by **k** and has the longest length.
Time Complexity: $O(n^2)$.

Method 2 (Efficient Approach): Create an array **mod_arr[]** where **mod_arr[i]** stores **(sum(arr[0]+arr[1]..+arr[i]) % k)**. Create a hash table having tuple as **(ele, idx)**, where **ele** represents an element of **mod_arr[]** and **idx** represents the element's index of first occurrence in **mod_arr[]**. Now, traverse **mod_arr[]** from i = 0 to n and follow the steps given below.

1. If mod_arr[i] == 0, then update **maxLen** = (i + 1).
2. Else if mod_arr[i] is not present in the hash table, then create tuple **(mod_arr[i], i)** in the hash table.
3. Else, get the value associated with **mod_arr[i]** in the hash table. Let this be **idx**.

4. If maxLen < (i – idx), then update **maxLen** = (i – idx).

Finally return **maxLen**.

C++

```
 // C++ implementation to find the longest subarray
// with sum divisible by k
#include <bits/stdc++.h>

using namespace std;

// function to find the longest subarray
// with sum divisible by k
int longSubarrWthSumDivByK(int arr[],
                           int n, int k)
{
    // unodered map 'um' implemented as
    // hash table
    unordered_map<int, int> um;

    // 'mod_arr[i]' stores (sum[0..i] % k)
    int mod_arr[n], max = 0;
    int curr_sum = 0;

    // traverse arr[] and build up the
    // array 'mod_arr[]'
    for (int i = 0; i < n; i++)
    {
        curr_sum += arr[i];

        // as the sum can be negative, taking modulo twice
        mod_arr[i] = ((curr_sum % k) + k) % k;
    }

    for (int i = 0; i < n; i++)
    {
        // if true then sum(0..i) is divisible
        // by k
        if (mod_arr[i] == 0)
            // update 'max'
            max = i + 1;

        // if value 'mod_arr[i]' not present in 'um'
        // then store it in 'um' with index of its
        // first occurrence
        else if (um.find(mod_arr[i]) == um.end())
            um[mod_arr[i]] = i;
```

```
        else
            // if true, then update 'max'
            if (max < (i - um[mod_arr[i]]))
                max = i - um[mod_arr[i]];
    }

    // required length of longest subarray with
    // sum divisible by 'k'
    return max;
}

// Driver program to test above
int main()
{
    int arr[] = {2, 7, 6, 1, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;

    cout << "Length = "
         << longSubarrWthSumDivByK(arr, n, k);

    return 0;
}
```

Java

```
 // Java implementation to find the longest
// subarray with sum divisible by k
import java.io.*;
import java.util.*;

class GfG {

    // function to find the longest subarray
    // with sum divisible by k
    static int longSubarrWthSumDivByK(int arr[],
                                      int n, int k)
    {
        // unodered map 'um' implemented as
        // hash table
        HashMap<Integer, Integer> um= new HashMap<Integer, Integer>();

        // 'mod_arr[i]' stores (sum[0..i] % k)
        int mod_arr[]= new int[n];
        int max = 0;
        int curr_sum = 0;
```

```
        // traverse arr[] and build up the
        // array 'mod_arr[]'
        for (int i = 0; i < n; i++)
        {
            curr_sum += arr[i];

            // as the sum can be negative,
            // taking modulo twice
            mod_arr[i] = ((curr_sum % k) + k) % k;
        }

        for (int i = 0; i < n; i++)
        {
            // if true then sum(0..i) is
            // divisible by k
            if (mod_arr[i] == 0)
                // update 'max'
                max = i + 1;

            // if value 'mod_arr[i]' not present in 'um'
            // then store it in 'um' with index of its
            // first occurrence
            else if (um.containsKey(mod_arr[i]) == false)
                um.put(mod_arr[i] , i);

            else
                // if true, then update 'max'
                if (max < (i - um.get(mod_arr[i])))
                    max = i - um.get(mod_arr[i]);
        }

        // required length of longest subarray with
        // sum divisible by 'k'
        return max;
    }

    public static void main (String[] args)
    {
        int arr[] = {2, 7, 6, 1, 4, 5};
        int n = arr.length;
        int k = 3;

        System.out.println("Length = "+
                           longSubarrWthSumDivByK(arr, n, k));

    }
}
```

```
// This code is contributed by Gitanjali.
```

Output:

```
Length = 3
```

Time Complexity: O(n).
Auxiliary Space: O(n^2).

Time complexity of this method can be improved by using an array of size equal to k for O(1) lookup since all elements would be less than k after using modulo operation on elements of input array.

Improved By : rex_wulf

Source

https://www.geeksforgeeks.org/longest-subarray-sum-divisible-k/

Chapter 183

Longest subsequence such that difference between adjacents is one | Set 2

Longest subsequence such that difference between adjacents is one | Set 2 - GeeksforGeeks

Given an array of size **n**. The task is to find the longest subsequence such that difference between adjacents is one. Time Complexity of O(n) is required.

Examples:

```
Input :  arr[] = {10, 9, 4, 5, 4, 8, 6}
Output :  3
As longest subsequences with difference 1 are, "10, 9, 8",
"4, 5, 4" and "4, 5, 6".

Input :  arr[] = {1, 2, 3, 2, 3, 7, 2, 1}
Output :  7
As longest consecutive sequence is "1, 2, 3, 2, 3, 2, 1".
```

Method 1: Previously an approach having time complexity of $O(n^2)$ have been discussed in this post.

Method 2 (Efficient Approach): The idea is to create a hash map having tuples in the form **(ele, len)**, where **len** denotes the length of the longest subsequence ending with the element **ele**. Now, for each element arr[i] we can find the length of the values arr[i]-1 and arr[i]+1 in the hash table and consider the maximum among them. Let this maximum value be **max**. Now, the length of longest subsequence ending with arr[i] would be **max+1**. Update this length along with the element arr[i] in the hash table. Finally, the element having the maximum length in the hash table gives the longest length subsequence.

```
 // C++ implementation to find longest subsequence
// such that difference between adjacents is one
#include <bits/stdc++.h>
using namespace std;

// function to find longest subsequence such
// that difference between adjacents is one
int longLenSub(int arr[], int n)
{
    // hash table to map the array element with the
    // length of the longest subsequence of which
    // it is a part of and is the last element of
    // that subsequence
    unordered_map<int, int> um;

    // to store the longest length subsequence
    int longLen = 0;

    // traverse the array elements
    for (int i=0; i<n; i++)
    {
        // initialize current length
        // for element arr[i] as 0
        int len = 0;

        // if 'arr[i]-1' is in 'um' and its length
        // of subsequence is greater than 'len'
        if (um.find(arr[i]-1) != um.end() &&
            len < um[arr[i]-1])
            len = um[arr[i]-1];

        // if 'arr[i]+1' is in 'um' and its length
        // of subsequence is greater than 'len'
        if (um.find(arr[i]+1) != um.end() &&
            len < um[arr[i]+1])
            len = um[arr[i]+1];

        // update arr[i] subsequence length in 'um'
        um[arr[i]] = len + 1;

        // update longest length
        if (longLen < um[arr[i]])
            longLen = um[arr[i]];
    }

    // required longest length subsequence
    return longLen;
}
```

```
// Driver program to test above
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 3, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Longest length subsequence = "
         << longLenSub(arr, n);
    return 0;
}
```

Output:

```
Longest length subsequence = 6
```

Time Complexity: O(n).
Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/longest-subsequence-difference-adjacents-one-set-2/

Chapter 184

Longest substring with count of 1s more than 0s

Longest substring with count of 1s more than 0s - GeeksforGeeks

Given a binary string find the longest substring which contains 1's more than 0's.

Examples:

```
Input : 1010
Output : 3
Substring 101 has 1 occuring more number of times than 0.

Input : 101100
Output : 5
Substring 10110 has 1 occuring more number of times than 0.
```

A **simple** solution is to one by one consider all the substrings and check if that substring has count of 1 more than 0. If count is more than compare its length with maximum length substring found till now. Time complexity of this solution is O(n^2).

An **efficient** solution is to use hashing. The idea is to find sum of string traversed until now. Add 1 to the result if current character is '1' else subtract 1. Now the problem reduces to finding largest subarray having sum greater than zero. To find largest subarray having sum greater than zero, we check the value of sum. If sum is greater than zero, then largest subarray with sum greater than zero is arr[0..i]. If sum is less than zero, then find size of subarray arr[j+1..i], where j is index upto which sum of subarray arr[0..j] is sum -1 and j < i and compare that size with largest subarray size found so far. To find index j, store values of sum for arr[0..j] in hash table for all 0 <= j <= i. There might be possibility that a given value of sum repeats. In that case store only first index for which that sum is obtained as it is required to get length of largest subarray and that is obtained from first index occurrence.

Below is the implementation of above approach:

```
 // CPP program to find largest substring
// having count of 1s more than count
// count of 0s.
#include <bits/stdc++.h>
using namespace std;

// Function to find longest substring
// having count of 1s more than count
// of 0s.
int findLongestSub(string bin)
{
    int n = bin.length(), i;

    // To store sum.
    int sum = 0;

    // To store first occurrence of each
    // sum value.
    unordered_map<int, int> prevSum;

    // To store maximum length.
    int maxlen = 0;

    // To store current substring length.
    int currlen;

    for (i = 0; i < n; i++) {

        // Add 1 if current character is 1
        // else subtract 1.
        if (bin[i] == '1')
            sum++;
        else
            sum--;

        // If sum is positive, then maximum
        // length substring is bin[0..i]
        if (sum > 0) {
            maxlen = i + 1;
        }

        // If sum is negative, then maximum
        // length substring is bin[j+1..i], where
        // sum of substring bin[0..j] is sum-1.
        else if (sum <= 0) {
            if (prevSum.find(sum - 1) != prevSum.end()) {
                currlen = i - prevSum[sum - 1];
                maxlen = max(maxlen, currlen);
```

```
            }
        }

        // Make entry for this sum value in hash
        // table if this value is not present.
        if (prevSum.find(sum) == prevSum.end())
            prevSum[sum] = i;
    }

    return maxlen;
}

// Driver code
int main()
{
    string bin = "1010";
    cout << findLongestSub(bin);
    return 0;
}
```

Output:

```
3
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Improved By : nik1996

Source

https://www.geeksforgeeks.org/longest-substring-with-count-of-1s-more-than-0s/

Chapter 185

MD5 hash in Java

MD5 hash in Java - GeeksforGeeks

To calculate cryptographic hashing value in Java, **MessageDigest** Class is used, under the package java.security.

MessagDigest Class provides following cryptographic hash function to find hash value of a text, they are:
1. MD5
2. SHA-1
3. SHA-256

This Algorithms are initialize in static method called **getInstance()**. After selecting the algorithm it calculate the **digest** value and return the results in byte array.

BigInteger class is used, which converts the resultant byte array into its **sign-magnitude representation**.

This representation converts into hex format to get the MessageDigest

Examples:

```
Input : hello world
Output : 5eb63bbbe01eeed093cb22bb8f5acdc3

Input : GeeksForGeeks
Output : e39b9c178b2c9be4e99b141d956c6ff6
```

```
 import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

// Java program to calculate MD5 hash value
public class MD5 {
```

```
    public static String getMd5(String input)
    {
        try {

            // Static getInstance method is called with hashing MD5
            MessageDigest md = MessageDigest.getInstance("MD5");

            // digest() method is called to calculate message digest
            //  of an input digest() return array of byte
            byte[] messageDigest = md.digest(input.getBytes());

            // Convert byte array into signum representation
            BigInteger no = new BigInteger(1, messageDigest);

            // Convert message digest into hex value
            String hashtext = no.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        }

        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    // Driver code
    public static void main(String args[]) throws NoSuchAlgorithmException
    {
        String s = "GeeksForGeeks";
        System.out.println("Your HashCode Generated by MD5 is: " + getMd5(s));
    }
}
```

Output:
Your HashCode Generated by MD5 is: e39b9c178b2c9be4e99b141d956c6ff6

References:
Java Docs for MessageDigest

Source

https://www.geeksforgeeks.org/md5-hash-in-java/

Chapter 186

Majority element in a circular array of 0's and 1's

Majority element in a circular array of 0's and 1's - GeeksforGeeks

Given a circular array containing only 0's and 1's, of size n where n = p*q (p and q are both odd integers). The task is to check if there is a way such that 1 will be in majority after applying the following operations:

1. Divide circular array into p subarrays each of size q.
2. In each subarray, the number which is in majority will get stored into array B.
3. Now, 1 will said to be in majority if it is in majority into array B.

Note: A number is in majority in an array if it occurs more than half times of the size of an array.

Examples:

> **Input:** p = 3, q = 3, array[] = {0, 0, 1, 1, 0, 1, 1, 0, 0}
> **Output:** Yes
> Assume index of the array from 1 to N, Since the array is circular so index N and 1 will be adjacent.
> Divide this circular array into subarray in this way :-
> {2, 3, 4}, {5, 6, 7} and {8, 9, 1}. [These are the index of elements]
> In {2, 3, 4}, 1 is in majority,
> in {5, 6, 7}, again 1 is in majority and
> In {8, 9, 1}, 0 is in majority.
> Now insert 1, 1, 0 into array B so array B = {1, 1, 0}
> In array B, 1 is the majority element so print Yes.
>
> **Input:** p = 3, q = 3, array[] = {1, 0, 0, 1, 1, 0, 1, 0, 0}
> **Output:** No
> No matter how you divide this circular subarray,
> 1 will not be in majority. Hence, the answer is No.

Approach:

1. First of all, iterate over the circular array and count the total number of 1's in each of p subarray(of size q).
2. Store this number into another array (of size p).
3. If in this case, 1 is in majority then, print yes.
4. Else take another set by moving the previous set's index 1 unit either increasing or decreasing it, and keep track of only those indices which are new in given set and update number of 1's in the array.
5. Repeat the 2nd and 3rd step.

We will repeat q times if there is no majority of 1 found until that, the answer would be NO, else (like in the previous example case) the answer would be 1.
Since at the maximum, we can repeat this process q times and each time we are tracking only two elements in each of p subarray.

Explanation:
In given example-1, we will divide circular subarray as [indices] => {1, 2, 3}, {4, 5, 6}, {7, 8, 9} and store number of 1's in another array which are [1, 2, 1] in subarrays respectively.

Taking another set in example 1 by increasing 1 unit so set will be {2, 3, 4}, {5, 6, 7}, {8, 9, 1} now in set 1 only change is inclusion of element 4 and removal of element 1, we will only track these so updated number of 1's will be 2, 2, 0.

Below is the implementation of above approach:

C++

```
 // C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to check if 1 is the majority
// element or not
void majority(bool a[], int p, int q, int size)
{
    // assuming starting and ending index of 1st subarray
    int start = 0, ends = q;

    // to store majority of p
    int arr[p];

    // subarrays each of size q ;
    int k = 0;

    // Loop to calculate total number
    // of 1's in subarray which will get
    // stored in array arr[]
```

```
while (k < p) {
    int one = 0;
    for (int j = start; j < ends; j++) {
        if (a[j] == 1) {
            one++;
        }
    }

    // starting index of next subarray
    start = ends;

    // ending index of next subarray
    ends = ends + q;

    // storing 1's
    arr[k] = one;
    k++;
}

start = 0;
ends = q;

// variable to keep a check
// if 1 is in majority or not
bool found = 0;

// In this case, we are repeating
// the task of calculating
// total number of 1's backward
while (ends > 0) {

    // to store the total number of 1's
    int dist_one = 0;

    // Check if 1 is in majority in
    // this subarray
    for (int i = 0; i < p; i++)
        if (arr[i] > q / 2)
            dist_one++;

    // If 1 is in majority return
    if (dist_one > p / 2) {
        found = 1;
        cout << "Yes" << endl;

        return;
    }
```

```
        // shifting starting index of
        // subarray by 1 unit leftwards
        start--;

        // shifting ending index of
        // subarray by 1 unit leftwards
        ends--;

        // to ensure it is a valid index
        // ( array is circular) -1 index means
        // last index of a circular array
        if (start < 0)
            start = size + start;

        int st = start, en = ends, l = 0;

        // now to track changes occur
        // due to shifting of the subarray
        while (en < size) {

            if (a[st % size] != a[en % size]) {

                // st refers to starting index of
                // new subarray and en refers to
                // last element of of same subarray
                // but in previous iteration
                if (a[st % size] == 1)
                    arr[l]++;

                else
                    arr[l]--;
            }
            l++;

            // now repeating the same
            // for other subarrays too
            st = (st + q);
            en = en + q;
        }
    }

    if (found == 0) {
        cout << "No" << endl;
    }
}

// Driver code
int main()
```

```
{
    int p = 3, q = 3;
    int n = p * q;

    bool a[] = { 0, 0, 1, 1, 0, 1, 1, 0, 0 };

    // circular array of given size
    majority(a, p, q, n);

    return 0;
}
```

Java

```
 // Java implementation of above approach

import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
// Function to check if 1 is the majority
// element or not
static void majority(int a[], int p, int q, int size)
{
    // assuming starting and ending index of 1st subarray
    int start = 0, ends = q;

    // to store majority of p
    int []arr = new int[p];

    // subarrays each of size q ;
    int k = 0;

    // Loop to calculate total number
    // of 1's in subarray which will get
    // stored in array arr[]
    while (k < p) {
        int one = 0;
        for (int j = start; j < ends; j++) {
            if (a[j] == 1) {
                one++;
            }
        }

        // starting index of next subarray
        start = ends;
```

```
    // ending index of next subarray
    ends = ends + q;

    // storing 1's
    arr[k] = one;
    k++;
}

start = 0;
ends = q;

// variable to keep a check
// if 1 is in majority or not
boolean  found = false;

// In this case, we are repeating
// the task of calculating
// total number of 1's backward
while (ends > 0) {

    // to store the total number of 1's
    int dist_one = 0;

    // Check if 1 is in majority in
    // this subarray
    for (int i = 0; i < p; i++)
        if (arr[i] > q / 2)
            dist_one++;

    // If 1 is in majority return
    if (dist_one > p / 2) {
        found = true;
        System.out.println( "Yes" );

        return;
    }

    // shifting starting index of
    // subarray by 1 unit leftwards
    start--;

    // shifting ending index of
    // subarray by 1 unit leftwards
    ends--;

    // to ensure it is a valid index
    // ( array is circular) -1 index means
```

```
        // last index of a circular array
        if (start < 0)
            start = size + start;

        int st = start, en = ends,l = 0;

        // now to track changes occur
        // due to shifting of the subarray
        while (en < size) {

            if (a[st % size] != a[en % size]) {

                // st refers to starting index of
                // new subarray and en refers to
                // last element of of same subarray
                // but in previous iteration
                if (a[st % size] == 1)
                    arr[l]++;

                else
                    arr[l]--;
            }
            l++;

            // now repeating the same
            // for other subarrays too
            st = (st + q);
            en = en + q;
        }
    }

    if (found == false ) {
        System.out.println( "No" );
    }
}

// Driver code
public static void main(String args[])
{
    int p = 3, q = 3;
    int n = p * q;

    int a[] = { 0, 0, 1, 1, 0, 1, 1, 0, 0 };

    // circular array of given size
    majority(a, p, q, n);
```

```
}
}

 // This code is Contributed by tufan_gupta2000
```

Output:

```
Yes
```

Improved By : tufan_gupta2000

Source

https://www.geeksforgeeks.org/majority-element-in-a-circular-array-of-0s-and-1s/

Chapter 187

Majority element in a linked list

Majority element in a linked list - GeeksforGeeks

Given a linked list, find majority element. An element is called **Majority element** if it appears more than or equal to **n/2** times where n is total number of nodes in the linked list.

Examples:

```
Input  : 1->2->3->4->5->1->1->1->NULL
Output : 1
Explanation  1 occurs 4 times
Input :10->23->11->9->54->NULL
Output :NO majority element
```

Method 1(simple)
Run two loops starting from head and count frequency of each element iteratively. Print the element whose frequency is greater than or equal to n/2. In this approach time complexity will be **O(n*n)** where n is the number of nodes in the linked list.

C++

```
 // C++ program to find majority element in
// a linked list
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};
```

```
/* Function to get the nth node from the
last of a linked list*/
int majority(struct Node* head)
{
    struct Node* p = head;

    int total_count = 0, max_count = 0, res = -1;
    while (p != NULL) {

        // Count all occurrences of p->data
        int count = 1;
        struct Node* q = p->next;
        while (q != NULL) {
            if (p->data == q->data)
               count++;
            q = q->next;
        }

        // Update max_count and res if count
        // is more than max_count
        if (count > max_count)
        {
            max_count = count;
            res = p->data;
        }

        p = p->next;
        total_count++;
    }

    if (max_count >= total_count/2)
        return res;

    // if we reach here it means no
    // majority element is present.
    // and we assume that all the
    // element are positive
    return -1;
}

void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
       (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
```

```
/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    // create linked
    push(&head, 1);
    push(&head, 1);
    push(&head, 1);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    int res = majority(head);

    if (res != (-1))
        cout << "Majority element is " << res;
    else
        cout << "No majority element";
    return 0;
}
```

Time Complexity O(n*n)

Method 2 Use hashing technique. We count frequency of each element and then we print the element whose frequency is n/2;

C++

```
 // CPP program to find majority element
// in the linked list using hashing
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* Function to get the nth node from the last
 of a linked list*/
int majority(struct Node* head)
{
```

```
    struct Node* p = head;

    // Storing elements and their frequencies
    // in a hash table.
    unordered_map<int, int> hash;

    int total_count = 0;
    while (p != NULL) {

        // increase every element
        // frequency by 1
        hash[p->data]++;

        p = p->next;

        total_count++;
    }

    // Check if frequency of any element
    // is more than or equal to total_count/2
    for (auto x : hash)
       if (x.second >= total_count/2)
           return x.first;

    // If we reach here means no majority element
    // is present. We assume that all the element
    // are positive
    return -1;
}

void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
       (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Driver program to test above function
int main()
{

    /* Start with the empty list */
    struct Node* head = NULL;
    push(&head, 1);
    push(&head, 1);
    push(&head, 1);
```

```
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    int res = majority(head);

    if (res != (-1))
        cout << "majority element is " << res;
    else
        cout << "NO majority elemenet";
    return 0;
}
```

Time Complexity O(n)

```
majority element is 1
```

Source

https://www.geeksforgeeks.org/majority-element-in-a-linked-list/

Chapter 188

Make two sets disjoint by removing minimum elements

Make two sets disjoint by removing minimum elements - GeeksforGeeks

Given two sets of integers as two arrays of size m and n. Find count of minimum numbers that should be removed from the sets so that both set become disjoint or don't contains any elements in common. We can remove elements from any set. We need to find minimum total elements to be removed.

Examples :

```
Input : set1[] = {20, 21, 22}
        set2[] = {22, 23, 24, 25}
Output : 1
We need to remove at least 1 element
which is 22 from set1 to make two
sets disjoint.

Input : set1[] = {20, 21, 22}
        set2[] = {22, 23, 24, 25, 20}
Output : 2

Input : set1[] = {6, 7}
        set2[] = {12, 13, 14, 15}
Output : 0
```

If we take a closer look at this problem, we can observe that the problem reduces to finding intersection of two sets.

A **simple solution** is iterate through every element of first set and for every element, check if it is present in second set. If present, increment count of elements to be removed.

C++

```
 // C++ simple program to find total elements
// to be removed to make two sets disjoint.
#include<bits/stdc++.h>
using namespace std;

// Function for find minimum elements to be
// removed from both sets so that they become
// disjoint
int makeDisjoint(int set1[], int set2[], int n, int m)
{
    int result = 0;
    for (int i=0; i<n; i++)
    {
        int j;
        for (j=0; j<m; j++)
            if (set1[i] == set2[j])
                break;

        if (j != m)
            result++;
    }

    return result;
}

// Driven code
int main()
{
    int set1[] = {20, 21, 22};
    int set2[] =  {22, 23, 24, 25, 20};
    int n = sizeof(set1)/sizeof(set1[0]);
    int m = sizeof(set2)/sizeof(set2[1]);
    cout << makeDisjoint(set1, set2, n, m);
    return 0;
}
```

Java

```
 // Java simple program to find
// total elements to be removed
// to make two sets disjoint.
import java.io.*;

public class GFG{
```

```
// Function for find minimum elements
// to be removed from both sets
// so that they become disjoint
static int makeDisjoint(int []set1,
                        int []set2,
                        int n,
                        int m)
{
    int result = 0;
    for (int i = 0; i < n; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (set1[i] == set2[j])
                break;

        if (j != m)
            result++;
    }

    return result;
}

    // Driver code
    static public void main (String[] args)
    {
    int []set1 = {20, 21, 22};
    int []set2 = {22, 23, 24, 25, 20};
    int n = set1.length;
    int m = set2.length;
    System.out.println(makeDisjoint(set1, set2,
                                    n, m));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# simple program to find
// total elements to be removed
// to make two sets disjoint.
using System;

public class GFG{

// Function for find minimum elements
// to be removed from both sets
```

```
// so that they become disjoint
static int makeDisjoint(int []set1,
                        int []set2,
                        int n,
                        int m)
{
    int result = 0;
    for (int i = 0; i < n; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (set1[i] == set2[j])
                break;

        if (j != m)
            result++;
    }

    return result;
}

    // Driver code
    static public void Main ()
    {
    int []set1 = {20, 21, 22};
    int []set2 = {22, 23, 24, 25, 20};
    int n = set1.Length;
    int m = set2.Length;
    Console.WriteLine(makeDisjoint(set1, set2,
                                   n, m));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP simple program to find
// total elements to be removed
// to make two sets disjoint.

// Function for find minimum
// elements to be removed from
// both sets so that they become disjoint
function makeDisjoint( $set1, $set2, $n, $m)
{
    $result = 0;
```

```
    for ( $i = 0; $i < $n; $i++)
    {
        $j;
        for ($j = 0; $j < $m; $j++)
            if ($set1[$i] == $set2[$j])
                break;

        if ($j != $m)
            $result++;
    }

    return $result;
}

// Driven code
$set1 = array(20, 21, 22);
$set2 = array(22, 23, 24, 25, 20);
$n = count($set1);
$m = count($set2);
echo makeDisjoint($set1, $set2, $n, $m);

// This code is contributed by anuj_67.
?>
```

Output :

```
2
```

Time complexity : O(m * n)
Auxiliary Space : O(1)

An **efficient solution** is to use hashing. We create an empty hash and insert all items of set 1 in it. Now iterate through set 2 and for every element, check if it is present is hash. If present, increment count of elements to be removed.

C++

```
 // C++ efficient program to find total elements
// to be removed to make two sets disjoint.
#include<bits/stdc++.h>
using namespace std;

// Function for find minimum elements to be
// removed from both sets so that they become
// disjoint
int makeDisjoint(int set1[], int set2[], int n, int m)
{
    // Store all elements of first array in a hash
```

```
    // table
    unordered_set <int> s;
    for (int i = 0; i < n; i++)
        s.insert(set1[i]);

    // We need to remove all those elements from
    // set2 which are in set1.
    int result = 0;
    for (int i = 0;  i < m; i++)
        if (s.find(set2[i]) != s.end())
            result++;

    return result;
}

// Driver code
int main()
{
    int set1[] = {20, 21, 22};
    int set2[] =  {22, 23, 24, 25, 20};
    int n = sizeof(set1)/sizeof(set1[0]);
    int m = sizeof(set2)/sizeof(set2[1]);
    cout << makeDisjoint(set1, set2, n, m);
    return 0;
}
```

Output :

```
2
```

Time complexity : O(m + n)
Auxiliary Space : O(m)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/make-two-sets-disjoint-removing-minimum-elements/

Chapter 189

Maximize elements using another array

Maximize elements using another array - GeeksforGeeks

Given two arrays with size n, maximize the first array by using the elements from the second array such that the new array formed contains n greatest but unique elements of both the arrays giving the second array priority (All elements of second array appear before first array). The order of appearance of elements is kept same in output as in input.

Examples:

> Input : arr1[] = {2, 4, 3}
> arr2[] = {5, 6, 1}
> Output : 5 6 4
> As 5, 6 and 4 are maximum elements from two arrays giving second array higher priority. Order of elements is same in output as in input.
>
> Input : arr1[] = {7, 4, 8, 0, 1}
> arr2[] = {9, 7, 2, 3, 6}
> Output : 9 7 6 4 8

Approach : We create an auxiliary array of size 2*n and store the elements of 2nd array in auxiliary array, and then we will store elements of 1st array in it. After that we will sort auxiliary array in decreasing order. To keep the order of elements according to input arrays we will use hash table. We will store 1st n largest unique elements of auxiliary array in hash table. Now we traverse the second array and store that elements of second array in auxiliary array that are present in hash table. Similarly we will traverse first array and store the elements that are present in hash table. In this way we get n unique and largest elements from both the arrays in auxiliary array while keeping the order of appearance of elements same.

Below is the implementation of above approach :

```
 // CPP program to print the maximum elements
// giving second array higher priority
#include <bits/stdc++.h>
using namespace std;

// Compare function used to sort array
// in decreasing order
bool compare(int a, int b)
{
    return a > b;
}

// Function to maximize array elements
void maximizeArray(int arr1[], int arr2[],
                                    int n)
{
    // auxiliary array arr3 to store
    // elements of arr1 & arr2
    int arr3[2*n], k = 0;
    for (int i = 0; i < n; i++)
        arr3[k++] = arr1[i];
    for (int i = 0; i < n; i++)
        arr3[k++] = arr2[i];

    // hash table to store n largest
    // unique elements
    unordered_set<int> hash;

    // sorting arr3 in decreasing order
    sort(arr3, arr3 + 2 * n, compare);

    // finding n largest unique elements
    // from arr3 and storing in hash
    int i = 0;
    while (hash.size() != n) {

        // if arr3 element not present in hash,
        // then store this element in hash
        if (hash.find(arr3[i]) == hash.end())
            hash.insert(arr3[i]);

        i++;
    }

    // store that elements of arr2 in arr3
    // that are present in hash
    k = 0;
    for (int i = 0; i < n; i++) {
```

```
        // if arr2 element is present in hash,
        // store it in arr3
        if (hash.find(arr2[i]) != hash.end()) {
            arr3[k++] = arr2[i];
            hash.erase(arr2[i]);
        }
    }

    // store that elements of arr1 in arr3
    // that are present in hash
    for (int i = 0; i < n; i++) {

        // if arr1 element is present in hash,
        // store it in arr3
        if (hash.find(arr1[i]) != hash.end()) {
            arr3[k++] = arr1[i];
            hash.erase(arr1[i]);
        }
    }

    // copying 1st n elements of arr3 to arr1
    for (int i = 0; i < n; i++)
        arr1[i] = arr3[i];
}

// Function to print array elements
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver Code
int main()
{
    int array1[] = { 7, 4, 8, 0, 1 };
    int array2[] = { 9, 7, 2, 3, 6 };
    int size = sizeof(array1) / sizeof(array1[0]);
    maximizeArray(array1, array2, size);
    printArray(array1, size);
}
```

Output:

```
9 7 6 4 8
```

Time complexity: O(n * log n).

Source

https://www.geeksforgeeks.org/maximize-elements-using-another-array/

Chapter 190

Maximum Unique Element in every subarray of size K

Maximum Unique Element in every subarray of size K - GeeksforGeeks

Given an array and an integer K. We need to find the maximum of every segment of length K which has no duplicates in that segment.

Examples:

```
Input : a[] = {1, 2, 2, 3, 3},
          K = 3.
Output : 1 3 2
For segment (1, 2, 2), Maximum = 1.
For segment (2, 2, 3), Maximum = 3.
For segment (2, 3, 3), Maximum = 2.

Input : a[] = {3, 3, 3, 4, 4, 2},
          K = 4.
Output : 4 Nothing 3
```

A **simple solution** is to run two loops. For every subarray find all distinct elements and print maximum unique element.

An **efficient solution** is to use sliding window technique. We maintain two structures in every window.
1) A hash table to store counts of all elements in current window.
2) A self balancing BST (implemented using set in C++ STL and TreeSet in Java). The idea is to quickly find maximum element and update maximum element.

We process first K-1 elements and store their counts in hash table. We also store unique elements in set. Now we one by one process last element of every window. If current element is unique, we add it to set. We also increase its count. After processing last element, we

print maximum from set. Before starting next iteration, we remove first element of previous window.

```
 // C++ code to calculate maximum unique
// element of every segment of array
#include <bits/stdc++.h>
using namespace std;

void find_max(int A[], int N, int K)
{
    // Storing counts of first K-1 elements
    // Also storing distinct elements.
    map<int, int> Count;
    for (int i = 0; i < K - 1; i++)
        Count[A[i]]++;
    set<int> Myset;
    for (auto x : Count)
        if (x.second == 1)
            Myset.insert(x.first);

    // Before every iteration of this loop,
    // we maintain that K-1 elements of current
    // window are processed.
    for (int i = K - 1; i < N; i++) {

        // Process K-th element of current window
        Count[A[i]]++;
        if (Count[A[i]] == 1)
            Myset.insert(A[i]);
        else
            Myset.erase(A[i]);

        // If there are no distinct
        // elements in current window
        if (Myset.size() == 0)
            printf("Nothing\n");

        // Set is ordered and last element
        // of set gives us maximum element.
        else
            printf("%d\n", *Myset.rbegin());

        // Remove first element of current
        // window before next iteration.
        int x = A[i - K + 1];
        Count[x]--;
        if (Count[x] == 1)
            Myset.insert(x);
```

```
        if (Count[x] == 0)
            Myset.erase(x);
    }
}

// Driver code
int main()
{
    int a[] = { 1, 2, 2, 3, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 3;
    find_max(a, n, k);
    return 0;
}
```

Output:

```
1
3
2
```

Time Complexity : O(N Log K)

Source

https://www.geeksforgeeks.org/maximum-unique-element-every-subarray-size-k/

Chapter 191

Maximum area rectangle by picking four sides from array

Maximum area rectangle by picking four sides from array - GeeksforGeeks

Given an array of n positive integers that represent lengths. Find out the maximum possible area whose four sides are picked from given array. Note that a rectangle can only be formed if there are two pairs of equal values in given array.

Examples:

```
Input : arr[] = {2, 1, 2, 5, 4, 4}
Output : 8
Explanation : Dimension will be 4 * 2

Input : arr[] = {2, 1, 3, 5, 4, 4}
Output : 0
Explanation : No rectangle possible
```

Method 1 (Sorting)
The task basically reduces to finding two pairs of equal values in array. If there are more than two pairs, then pick the two pairs with maximum values. A simple solution is to do following.
1) Sort the given array.
2) Traverse array from largest to smallest value and return two pairs with maximum values.

C++

```
 // CPP program for finding maximum area possible
// of a rectangle
#include <bits/stdc++.h>
using namespace std;
```

```
// function for finding max area
int findArea(int arr[], int n)
{
    // sort array in non-increasing order
    sort(arr, arr + n, greater<int>());

    // Initialize two sides of rectangle
    int dimension[2] = { 0, 0 };

    // traverse through array
    for (int i = 0, j = 0; i < n - 1 && j < 2; i++)

        // if any element occurs twice
        // store that as dimension
        if (arr[i] == arr[i + 1])
            dimension[j++] = arr[i++];

    // return the product of dimensions
    return (dimension[0] * dimension[1]);
}

// driver function
int main()
{
    int arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findArea(arr, n);
    return 0;
}
```

Java

```
 // Java program for finding maximum area
// possible of a rectangle
import java.util.Arrays;
import java.util.Collections;

public class GFG
{
    // function for finding max area
    static int findArea(Integer arr[], int n)
    {
        // sort array in non-increasing order
        Arrays.sort(arr, Collections.reverseOrder());

        // Initialize two sides of rectangle
        int[] dimension = { 0, 0 };
```

```
        // traverse through array
        for (int i = 0, j = 0; i < n - 1 && j < 2;
                                            i++)

            // if any element occurs twice
            // store that as dimension
            if (arr[i] == arr[i + 1])
                dimension[j++] = arr[i++];

        // return the product of dimensions
        return (dimension[0] * dimension[1]);
    }

    // driver function
    public static void main(String args[])
    {
        Integer arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
        int n = arr.length;
        System.out.println(findArea(arr, n));
    }
}
// This code is contributed by Sumit Ghosh
```

Python3

```
 # Python3 program for finding
# maximum area possible of
# a rectangle

# function for finding
# max area
def findArea(arr, n):

    # sort array in
    # non-increasing order
    arr.sort(reverse = True)

    # Initialize two
    # sides of rectangle
    dimension = [0, 0]

    # traverse through array
    i = 0
    j = 0
    while(i < n - 1 and j < 2):

        # if any element occurs twice
```

```
        # store that as dimension
        if (arr[i] == arr[i + 1]):
            dimension[j] = arr[i]
            j += 1
            i += 1
        i += 1

    # return the product
    # of dimensions
    return (dimension[0] *
            dimension[1])

# Driver code
arr = [4, 2, 1, 4, 6, 6, 2, 5]
n = len(arr)
print(findArea(arr, n))

# This code is contributed
# by Smitha
```

PHP

```
 <?php
// PHP program for finding maximum area possible
// of a rectangle

// function for finding max area
function findArea($arr, $n)
{

    // sort array in non-
    // increasing order
    rsort($arr);

    // Initialize two sides
    // of rectangle
    $dimension = array( 0, 0 );

    // traverse through array
    for( $i = 0, $j = 0; $i < $n - 1 &&
                            $j < 2; $i++)

        // if any element occurs twice
        // store that as dimension
        if ($arr[$i] == $arr[$i + 1])
            $dimension[$j++] = $arr[$i++];

    // return the product
```

```
    // of dimensions
    return ($dimension[0] *
            $dimension[1]);
}

    // Driver Code
    $arr = array(4, 2, 1, 4, 6, 6, 2, 5);
    $n =count($arr);
    echo findArea($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
24
```

Time Complexity : O(n Log n)

Method 2 (Hashing)
The idea is to insert all first occurrences of elements in a hash set. For second occurrences, keep track of maximum two values.

C++

```
 // CPP program for finding maximum area possible
// of a rectangle
#include <bits/stdc++.h>
using namespace std;

// function for finding max area
int findArea(int arr[], int n)
{
    unordered_set<int> s;

    // traverse through array
    int first = 0, second = 0;
    for (int i = 0; i < n; i++) {

        // If this is first occurrence of arr[i],
        // simply insert and continue
        if (s.find(arr[i]) == s.end()) {
            s.insert(arr[i]);
            continue;
        }
```

```
        // If this is second (or more) occurrence,
        // update first and second maximum values.
        if (arr[i] > first) {
            second = first;
            first = arr[i];
        } else if (arr[i] > second)
            second = arr[i];
    }

    // return the product of dimensions
    return (first * second);
}

// driver function
int main()
{
    int arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findArea(arr, n);
    return 0;
}
```

Java

```
 // Java program for finding maximum
// area possible of a rectangle
import java.util.HashSet;
import java.util.Set;

public class GFG
{
    // function for finding max area
    static int findArea(int arr[], int n)
    {
        //unordered_set<int> s;

        Set<Integer> s = new HashSet<>();

        // traverse through array
        int first = 0, second = 0;
        for (int i = 0; i < n; i++) {

            // If this is first occurrence of
            // arr[i], simply insert and continue
            if (!s.contains(arr[i])) {
                s.add(arr[i]);
                continue;
```

```
            }

            // If this is second (or more)
            // occurrence, update first and
            // second maximum values.
            if (arr[i] > first) {
                second = first;
                first = arr[i];
            } else if (arr[i] > second)
                second = arr[i];
        }

        // return the product of dimensions
        return (first * second);
    }

    // driver function
    public static void main(String args[])
    {
        int arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
        int n = arr.length;
        System.out.println(findArea(arr, n));
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
24
```

Time Complexity : O(n)

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/maximum-area-rectangle-picking-four-sides-array/

Chapter 192

Maximum array from two given arrays keeping order same

Maximum array from two given arrays keeping order same - GeeksforGeeks

Given two same sized arrays A[] and B[] (both arrays contain distinct elements individually but may have some common elements), task is to form a third (or result) array of same size. The result array should have maximum n elements from both array. It should have chosen elements of A[] first, then chosen elements of B[] in same order as they appear in original arrays. If there are common elements, then only one element should be present in res[] and priority should be given to A[].

Examples:

```
Input :  A[] =  [ 9 7 2 3 6 ]
         B[] =  [ 7 4 8 0 1 ]
Output : res[] = [9 7 6 4 8]
res[] has maximum n elements of both A[]
and B[] such that elements of A[] appear
first (in same order), then elements of B[].
Also 7 is common and priority is given to
A's 7.

Input :  A[] = [ 6 7 5 3 ]
         B[] = [ 5 6 2 9 ]
Output : res[] = [ 6 7 5 9 ]
```

1) Create copies of both arrays and sort the copies in decreasing order.
2) Use a hash to pick unique n maximum elements of both arrays, giving priority to A[].
3) Initialize result array as empty.
4) Traverse through A[], copy those elements of A[] that are present in the hash. This is done to keep order of elements same.

5) Repeat step 4 for B[]. This time we only consider those elements that are not present in A[] (Do not appear twice in hash).

Below c++ implementation of above idea.

```
 // Make a set of maximum elements from two
// arrays A[] and B[]
#include <bits/stdc++.h>
using namespace std;

void maximizeTheFirstArray(int A[], int B[],
                                    int n)
{
    // Create copies of A[] and B[] and sort
    // the copies in descending order.
    vector<int> temp1(A, A+n);
    vector<int> temp2(B, B+n);
    sort(temp1.begin(), temp1.end(), greater<int>());
    sort(temp2.begin(), temp2.end(), greater<int>());

    // Put maximum n distinct elements of
    // both sorted arrays in a map.
    unordered_map<int, int> m;
    int i = 0, j = 0;
    while (m.size() < n)
    {
         if (temp1[i] >= temp2[j])
         {
            m[temp1[i]]++;
            i++;
         }
         else
         {
            m[temp2[j]]++;
            j++;
         }
    }

    // Copy elements of A[] to that
    // are present in hash m.
    vector<int> res;
    for (int i = 0; i < n; i++)
        if (m.find(A[i]) != m.end())
           res.push_back(A[i]);

    // Copy elements of B[] to that
    // are present in hash m. This time
    // we also check if the element did
    // not appear twice.
```

```
    for (int i = 0; i < n; i++)
        if (m.find(B[i]) != m.end() &&
            m[B[i]] == 1)
           res.push_back(B[i]);

    // print result
    for (int i = 0; i < n; i++)
        cout << res[i] << " ";
}

// driver program
int main()
{
    int A[] = { 9, 7, 2, 3, 6 };
    int B[] = { 7, 4, 8, 0, 1 };
    int n = sizeof(A) / sizeof(A[0]);
    maximizeTheFirstArray(A, B, n);
    return 0;
}
```

Output:

```
 9 7 6 4 8
```

Time complexity: O(n Log n)

Source

https://www.geeksforgeeks.org/maximum-array-from-two-given-arrays-keeping-order-same/

Chapter 193

Maximum consecutive numbers present in an array

Maximum consecutive numbers present in an array - GeeksforGeeks

Find the length of maximum number of consecutive numbers jumbled up in an array.

Examples:

```
Input : arr[] = {1, 94, 93, 1000, 5, 92, 78};
Output : 3
The largest set of consecutive elements is
92, 93, 94

Input  : arr[] = {1, 5, 92, 4, 78, 6, 7};
Output : 4
The largest set of consecutive elements is
4, 5, 6, 7
```

The idea is to use hashing. We traverse through the array and for every element, we check if it is the starting element of its sequence. If yes then by incrementing its value we search the set and increment the length. By repeating this for all elements, we can find the lengths of all consecutive sets in array. Finally we return length of the largest set.

```
 // CPP program to find largest consecutive numbers
// present in arr[].
#include <bits/stdc++.h>
using namespace std;

int findLongestConseqSubseq(int arr[], int n)
{
```

```
    /* We insert all the array elements into
       unordered set. */
    unordered_set<int> S;
    for (int i = 0; i < n; i++)
        S.insert(arr[i]);

    // check each possible sequence from the start
    // then update optimal length
    int ans = 0;
    for (int i = 0; i < n; i++) {

        // if current element is the starting
        // element of a sequence
        if (S.find(arr[i] - 1) == S.end()) {

            // Then check for next elements in the
            // sequence
            int j = arr[i];

            // increment the value of array element
            // and repeat search in the set
            while (S.find(j) != S.end())
                j++;

            // Update  optimal length if this length
            // is more. To get the length as it is
            // incremented one by one
            ans = max(ans, j - arr[i]);
        }
    }
    return ans;
}

// Driver code
int main()
{
    int arr[] = { 1, 94, 93, 1000, 5, 92, 78 };
    int n = sizeof(arr) / sizeof(int);
    cout << findLongestConseqSubseq(arr, n) << endl;
    return 0;
}
```

Output:

```
3
```

Time complexity : O(n)

Source

https://www.geeksforgeeks.org/maximum-consecutive-numbers-present-array/

Chapter 194

Maximum difference between first and last indexes of an element in array

Maximum difference between first and last indexes of an element in array - GeeksforGeeks

Given an array of n integers. The task is to find the difference of first and last index of each distinct element so as to maximize the difference.

Examples:

```
Input : {2, 1, 3, 4, 2, 1, 5, 1, 7}
Output : 6
Element 1 has its first index = 1
and last index = 7
Difference = 7 - 1 = 6
Other elements have a smaller first and last
index difference

Input : {2, 2, 1, 1, 8, 8, 3, 5, 3}
Output : 2
Maximum difference is for indexes of element 3.
```

A **simple approach** is to run two loops and find the difference for each element and accordingly update the **max_diff**. It has a time complexity of $O(n^2)$ and the approach also needs to keep track of the elements that have been visited so that difference for them is not calculated unnecessarily.

An **efficient approach** uses hashing. It has the following steps.

1. Traverse the input array from left to right.

2. For each distinct element map its first and last index in the hash table.
3. Traverse the hash table and calculate the first and last index difference for each element.
4. Accordingly update the **max_diff**.

In the following implementation unordered_map has been used for hashing as the range of integers is not known.

C++

```
 // C++ implementation to find the maximum difference
// of first and last index of array elements
#include <bits/stdc++.h>

using namespace std;

// function to find the
// maximum difference
int maxDifference(int arr[], int n)
{
    // structure to store first and last
    // index of each distinct element
    struct index
    {
        int f, l;
    };

    // maps each element to its
    // 'index' structure
    unordered_map<int, index> um;

    for (int i=0; i<n; i++)
    {
        // storing first index
        if (um.find(arr[i]) == um.end())
            um[arr[i]].f = i;

        // storing last index
        um[arr[i]].l = i;
    }

    int diff, max_diff = INT_MIN;

    unordered_map<int, index>::iterator itr;

    // traversing 'um'
    for (itr=um.begin(); itr != um.end(); itr++)
```

```
    {
        // difference of last and first index
        // of each element
        diff = (itr->second).l - (itr->second).f;

        // update 'max_dff'
        if (max_diff < diff)
            max_diff = diff;
    }

    // required maximum difference
    return max_diff;
}


// Driver program to test above
int main()
{
    int arr[] = {2, 1, 3, 4, 2, 1, 5, 1, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = "
         <<maxDifference(arr, n);
    return 0;
}
```

Java

```
 // Java implementation to find the maximum difference
// of first and last index of array elements
import java.util.HashMap;
import java.util.Map;

public class MaxDiffIndexHashing {

    static class Element {
        int first;
        int second;

        public Element() {
            super();
        }

        public Element(int first, int second) {
            super();
            this.first = first;
            this.second = second;
        }
    }
```

```
    public static void main(String[] args) {

        int arr[]={2, 1, 3, 4, 2, 1, 5, 1, 7};
        System.out.println("Maximum Difference= "+ maxDiffIndices(arr));
    }

    private static int maxDiffIndices(int[] arr) {
        int n = arr.length;
        int maxDiffIndex = 0;
        Map<Integer, Element> map = new HashMap<Integer, Element>();

        for (int i = 0; i < n; i++) {
            if (map.containsKey(arr[i])) {
                Element e = map.get(arr[i]);
                e.second = i;
            } else {
                Element e = new Element();
                e.first = i;
                map.put(arr[i], e);
            }

        }

        for (Map.Entry<Integer, Element> entry : map.entrySet()) {
            Element e = entry.getValue();
            if ((e.second - e.first) > maxDiffIndex)
                maxDiffIndex = e.second - e.first;
        }

        return maxDiffIndex;
    }

}
```

Output:

```
Maximum Difference = 6
```

Time Complexity: O(n)

Improved By : Vardhan Vishnu

Source

https://www.geeksforgeeks.org/maximum-difference-first-last-indexes-element-array/

Chapter 195

Maximum difference between frequency of two elements such that element having greater frequency is also greater

Maximum difference between frequency of two elements such that element having greater frequency is also greater - GeeksforGeeks

Given an array of **n** positive integers with many repeating elements. The task is to find maximum difference between the frequency of any two different elements, such that the element with greater frequency is also greater in value than the second integer.

Examples:

```
Input :  arr[] = { 3, 1, 3, 2, 3, 2 }.
Output : 2
Frequency of 3 = 3.
Frequency of 2 = 2.
Frequency of 1 = 1.
Here difference of frequency of element 3 and 1 is = 3 - 1 = 2.
Also 3 > 1.
```

Method 1 (Use Hashing):

The naive approach can be, find the frequency of each element and for each element find the element having lesser value and lesser frequency than the current element.

Below is C++ implementation of this approach:

```
// C++ program to find maximum difference
```

```
// between frequency of any two element
// such that element with greater frequency
// is also greater in value.
#include<bits/stdc++.h>
using namespace std;

// Return the maximum difference between
// frequencies of any two elements such that
// element with greater frequency is also
// greater in value.
int maxdiff(int arr[], int n)
{
    unordered_map<int, int> freq;

    // Finding the frequency of each element.
    for (int i = 0; i < n; i++)
        freq[arr[i]]++;

    int ans = 0;
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            // finding difference such that element
            // having greater frequency is also
            // greater in value.
            if (freq[arr[i]] > freq[arr[j]] &&
                arr[i] > arr[j] )
                ans = max(ans, freq[arr[i]]-freq[arr[j]]);
            else if (freq[arr[i]] < freq[arr[j]] &&
                      arr[i] < arr[j] )
                ans = max(ans, freq[arr[j]]-freq[arr[i]]);
        }
    }

    return ans;
}

// Driven Program
int main()
{
    int arr[] = { 3, 1, 3, 2, 3, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << maxdiff(arr, n) << endl;
    return 0;
}
```

Output:

```
2
```

Time Complexity: $O(n^2)$.

Method 2 (Use Hashing and Sorting):

The idea is to find all the distinct elements and store in an array, say dist[ ]. Sort the distinct element array dist[] in increasing order. Now for any distinct element at index i, for all index j such that $i > j > 0$, find the element between index 0 to i-1 having minimum frequency. We can find frequency of an element in same way as method 1, i.e., storing frequencies in a hash table.
So do this for all i and find the maximum difference. To find the minimum frequency for all i maintain a prefix minimum.

Below is C++ representation of this approach:

```
 // Efficient C++ program to find maximum
// difference between frequency of any two
// elements such that element with greater
// frequency is also greater in value.
#include<bits/stdc++.h>
using namespace std;

// Return the maximum difference between
// frequencies of any two elements such that
// element with greater frequency is also
// greater in value.
int maxdiff(int arr[], int n)
{
    unordered_map<int, int> freq;

    int dist[n];

    // Finding the frequency of each element.
    int j = 0;
    for (int i = 0; i < n; i++)
    {
        if (freq.find(arr[i]) == freq.end())
            dist[j++] = arr[i];

        freq[arr[i]]++;
    }
```

```
    // Sorting the distinct element
    sort(dist, dist + j);

    int min_freq = n+1;

    // Iterate through all sorted distinct elements.
    // For each distinct element, maintaining the
    // element with minimum frequency than that
    // element and also finding the maximum
    // frequency difference
    int ans = 0;
    for (int i=0; i<j; i++)
    {
        int cur_freq = freq[dist[i]];
        ans = max(ans, cur_freq - min_freq);
        min_freq = min(min_freq, cur_freq);
    }

    return ans;
}

// Driven Program
int main()
{
    int arr[] = { 3, 1, 3, 2, 3, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << maxdiff(arr, n) << endl;
    return 0;
}
```

Output:

```
2
```

Time Complexity : O(n log n).

Source

https://www.geeksforgeeks.org/maximum-difference-between-frequency-of-two-elements-such-that-element-having-

Chapter 196

Maximum distance between two occurrences of same element in array

Maximum distance between two occurrences of same element in array - GeeksforGeeks

Given an array with repeated elements, the task is to find the maximum distance two occurrences of an element.

Examples:

```
Input : arr[] = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2}
Output: 10
// maximum distance for 2 is 11-1 = 10
// maximum distance for 1 is 4-2 = 2
// maximum distance for 4 is 10-5 = 5
```

A **simple solution** for this problem is to one by one pick each element from array and find its **first** and **last** occurence in array and take difference of first and last occurence for maximum distance. Time complexity for this approach is $O(n^2)$.

An **efficient solution** for this problem is to use hashing. The idea is to traverse input array and store index of first occurrence in a hash map. For every other occurrence, find the difference between index and the first index stored in hash map. If difference is more than result so far, then update the result.

Below are implementations of the idea. The C++ implementation uses unordered_map in C++.

C++

```
// C++ program to find maximum distance between two
```

```
// same occurrences of a number.
#include<bits/stdc++.h>
using namespace std;

// Function to find maximum distance between equal elements
int maxDistance(int arr[], int n)
{
    // Used to store element to first index mapping
    unordered_map<int, int> mp;

    // Traverse elements and find maximum distance between
    // same occurrences with the help of map.
    int max_dist = 0;
    for (int i=0; i<n; i++)
    {
        // If this is first occurrence of element, insert its
        // index in map
        if (mp.find(arr[i]) == mp.end())
            mp[arr[i]] = i;

        // Else update max distance
        else
            max_dist = max(max_dist, i - mp[arr[i]]);
    }

    return max_dist;
}

// Driver program to run the case
int main()
{
    int arr[] = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << maxDistance(arr, n);
    return 0;
}
```

Python

```
 # Python program to find maximum distance between two
# same occurrences of a number.

# Function to find maximum distance between equal elements
def maxDistance(arr, n):

    # Used to store element to first index mapping
    mp = {}
```

```
    # Traverse elements and find maximum distance between
    # same occurrences with the help of map.
    maxDict = 0
    for i in range(n):

        # If this is first occurrence of element, insert its
        # index in map
        if arr[i] not in mp.keys():
            mp[arr[i]] = i

        # Else update max distance
        else:
            maxDict = max(maxDict, i-mp[arr[i]])

    return maxDict

# Driver Program
if __name__=='__main__':
    arr = [3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2]
    n = len(arr)
    print maxDistance(arr, n)

# Contributed By: Harshit Sidhwa
```

Output:

```
10
```

Time complexity : O(n) under the assumption that unordered_map's search and insert operations take O(1) time.

Source

https://www.geeksforgeeks.org/maximum-distance-two-occurrences-element-array/

Chapter 197

Maximum distinct elements after removing k elements

Maximum distinct elements after removing k elements - GeeksforGeeks

Given an array **arr**[] containing **n** elements. The problem is to find maximum number of distinct elements after removing **k** elements from the array.
Note: 1 <= k <= n.

Examples:

```
Input : arr[] = {5, 7, 5, 5, 1, 2, 2}, k = 3
Output : 4
Remove 2 occurrences of element 5 and
1 occurrence of element 2.

Input : arr[] = {1, 2, 3, 4, 5, 6, 7}, k = 5
Output : 2
```

Approach: Following are the steps:

1. Create a hash table to store the frequency of each element.
2. Insert frequency of each element in a max heap.
3. Now, perform the following operation **k** times. Remove an element from the max heap. Decrement its value by 1. After this if element is not equal to 0, then again push the element in the max heap.
4. After the completion of step 3, the number of elements in the max heap is the required answer.

```
 // C++ implementation to find maximum distinct
// elements after removing k elements
```

```
#include <bits/stdc++.h>
using namespace std;

// function to find maximum distinct elements
// after removing k elements
int maxDistinctNum(int arr[], int n, int k)
{
    // 'um' implemented as hash table to store
    // frequency of each element
    unordered_map<int, int> um;

    // priority_queue 'pq' implemented as
    // max heap
    priority_queue<int> pq;

    // storing frequency of each element in 'um'
    for (int i = 0; i < n; i++)
        um[arr[i]]++;

    // inserting frequency of each element in 'pq'
    for (auto it = um.begin(); it != um.end(); it++)
        pq.push(it->second);

    while (k--) {

        // get the top element of 'pq'
        int temp = pq.top();

        // remove top element from 'pq'
        pq.pop();

        // decrement the popped element by 1
        temp--;

        // if true, then push the element in 'pq'
        if (temp != 0)
            pq.push(temp);
    }

    // required maximum distinct elements
    return ((int)pq.size());
}

// Driver program to test above
int main()
{
    int arr[] = { 5, 7, 5, 5, 1, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    int k = 3;
    cout << "Maximum distinct elements = "
         << maxDistinctNum(arr, n, k);
    return 0;
}
```

Output:

```
Maximum distinct elements = 4
```

Time Complexity: O(k*logd), where **d** is the number of distinct elements in the given array.

Source

https://www.geeksforgeeks.org/maximum-distinct-elements-removing-k-elements/

Chapter 198

Maximum distinct lowercase alphabets between two uppercase

Maximum distinct lowercase alphabets between two uppercase - GeeksforGeeks

Given a string containing alphabets in lowercase and uppercase, find the maximum count of distinct lowercase alphabets present between two uppercase alphabets.

Examples :

```
Input : zACaAbbaazzC
Output : The maximum count = 3

Input : edxedxxxCQiIVmYEUtLi
Output : The maximum count = 1
```

Method 1 (Using Character Count Array):

- Declare an array of size 26 where each index of the array represents a character in English alphabet
- Iterate the string over its complete length
- For each lowercase character, increment the index of the corresponding array by 1.
- For each uppercase character, iterate the array and count the number of positions having value greater than zero.
- If this count is greater than the maximum count, update the maximum counter and **initialize the array by 0**.

Below is the implementation of the above method.

C++

```
 // CPP Program to find maximum
// lowercase alphabets present
// between two uppercase alphabets
#include <bits/stdc++.h>
using namespace std;

#define MAX_CHAR 26

// Function which computes the
// maximum number of distinct
// lowercase alphabets between
// two uppercase alphabets
int maxLower(string str)
{
    int n = str.length();

    // Ignoring lowercase characters in the
    // beginning.
    int i = 0;
    for (; i < n; i++) {
        if (str[i] >= 'A' && str[i] <= 'Z') {
            i++;
            break;
        }
    }

    // We start from next of first capital letter
    // and traverse through remaining character.
    int maxCount = 0;
    int count[MAX_CHAR] = { 0 };
    for (; i < n; i++) {

        // If character is in uppercase,
        if (str[i] >= 'A' && str[i] <= 'Z') {

            // Count all distinct lower case
            // characters
            int currCount = 0;
            for (int j = 0; j < MAX_CHAR; j++)
                if (count[j] > 0)
                    currCount++;

            // Update maximum count
            maxCount = max(maxCount, currCount);

            // Reset count array
            memset(count, 0, sizeof(count));
        }
```

```
        // If character is in lowercase
        if (str[i] >= 'a' && str[i] <= 'z')
            count[str[i] - 'a']++;
    }

    return maxCount;
}

// Driver function
int main()
{
    string str = "zACaAbbaazzC";
    cout << maxLower(str);
    return 0;
}
```

Python3

```
 # Python3 Program to find maximum
# lowercase alphabets present
# between two uppercase alphabets

MAX_CHAR = 26

# Function which computes the
# maximum number of distinct
# lowercase alphabets between
# two uppercase alphabets
def maxLower(str):
    n = len(str)

    # Ignoring lowercase characters
    # in the beginning.
    i = 0
    for i in range(n):
        if str[i] >= 'A' and str[i] <= 'Z':
            i += 1
            break

    # We start from next of first capital
    # letter and traverse through
    # remaining character.
    maxCount = 0
    count = []
    for j in range(MAX_CHAR):
        count.append(0)
```

```
    for j in range(i, n):

        # If character is in uppercase,
        if str[j] >= 'A' and str[j] <= 'Z':

            # Count all distinct lower
            # case characters
            currCount = 0
            for k in range(MAX_CHAR):
                if count[k] > 0:
                    currCount += 1

            # Update maximum count
            maxCount = max(maxCount, currCount)

            # Reset count array
            for y in count:
                y = 0

        # If character is in lowercase
        if str[j] >= 'a' and str[j] <= 'z':
            count[ord(str[j]) - ord('a')] += 1

    return maxCount

# Driver function
str = "zACaAbbaazzC";
print(maxLower(str))

# This code is contributed by Upendra Bartwal
```

Output:

```
3
```

Time Complexity : O(n).

Method 2 (Using Hash Table): In this method, we extensively use the C++ STL container unordered_set.

Below is the implementation of above method :

```
 // CPP Program to find maximum
// lowercase alphabets present
// between two uppercase alphabets
#include <bits/stdc++.h>
using namespace std;
```

```
// Function which computes the
// maximum number of distinct
// lowercase alphabets between
// two uppercase alphabets
int maxLower(string str)
{
    int n = str.length();

    // Ignoring lowercase characters in the
    // beginning.
    int i = 0;
    for (; i < n; i++) {
        if (str[i] >= 'A' && str[i] <= 'Z') {
            i++;
            break;
        }
    }

    // We start from next of first capital letter
    // and traverse through remaining character.
    int maxCount = 0;
    unordered_set<int> s;
    for (; i < n; i++) {

        // If character is in uppercase,
        if (str[i] >= 'A' && str[i] <= 'Z') {

            // Update maximum count if lowercase
            // character before this is more.
            maxCount = max(maxCount, (int)s.size());

            // clear the set
            s.clear();
        }

        // If character is in lowercase
        if (str[i] >= 'a' && str[i] <= 'z')
            s.insert(str[i]);
    }

    return maxCount;
}

// Driver function
int main()
{
    string str = "zACaAbbaazzC";
```

```
    cout << maxLower(str);
    return 0;
}
```

Output:

```
3
```

Time complexity : $O(n)$

Improved By : aganjali10

Source

https://www.geeksforgeeks.org/maximum-distinct-lowercase-alphabets-two-uppercase/

Chapter 199

Maximum length subsequence possible of the form R^N K^N

Maximum length subsequence possible of the form R^N K^N - GeeksforGeeks

Given a string containing only two characters i.e. R and K (like RRKRRKKKKK). The task is to find the maximum value of N for a subsequence possible of the form R—N times and then K—N times (i.e. of the form R^N K^N).

Note: String of k should be started after the string of R i.e. first k that would be considered for 'K' string must occur after the last R of the 'R' string in the given string. Also, the length of the resulting subsequence will be 2*N.

Examples:

> **Input**: RRRKRRKKRRKKK
> **Output**: 5
> If we take R's at indexes 0, 1, 2, 4, 5 and K's at indexes 6, 7, 10, 11, 12
> then we get a maximum subsequence of the form R^N K^N, where N = 5.
>
> **Input**: KKKKRRRRK
> **Output**: 1
> If we take R at index 4(or 5 or 6 or 7) and K at index 8
> then we get the desired subsequence with N = 1.

Approach:

1. Calculate the number of R's before a K .
2. Calculate the number of K's after a K, including that K.
3. Store them in a table with a number of R's in table[x][0] and a number of K's in table[x][1].
4. Minimum of the two gives the value of n for each K and we will the return the maximum n.

Below is the implementation of the above approach:

Java

```
 // Java program to find the maximum
// length of a substring of form R^nK^n
public class gfg {

    // function to calculate the maximum
    // length of substring of the form R^nK^n
    int find(String s)
    {
        int max = 0, i, j = 0, countk = 0, countr = 0;
        int table[][] = new int[s.length()][2];

        // Count no. Of R's before a K
        for (i = 0; i < s.length(); i++) {
            if (s.charAt(i) == 'R')
                countr++;
            else
                table[j++][0] = countr;
        }
        j--;

        // Count no. Of K's after a K
        for (i = s.length() - 1; i >= 0; i--) {
            if (s.charAt(i) == 'K') {
                countk++;
                table[j--][1] = countk;
            }

            // Update maximum length
            if (Math.min(table[j + 1][0], table[j + 1][1]) > max)
                max = Math.min(table[j + 1][0], table[j + 1][1]);
        }

        return max;
    }

    // Driver code
    public static void main(String srgs[])
    {
        String s = "RKRRRKKRRKKKKRR";
        gfg ob = new gfg();
        int n = ob.find(s);
        System.out.println(n);
    }
}
```

C#

```
 // C# program to find the maximum
// length of a substring of
// form R^nK^n
using System;

class GFG
{

// function to calculate the
// maximum length of substring
// of the form R^nK^n
static int find(String s)
{
    int max = 0, i, j = 0,
        countk = 0, countr = 0;
    int [,]table= new int[s.Length, 2];

    // Count no. Of R's before a K
    for (i = 0; i < s.Length; i++)
    {
        if (s[i] == 'R')
            countr++;
        else
            table[(j++),0] = countr;
    }
    j--;

    // Count no. Of K's after a K
    for (i = s.Length - 1; i >= 0; i--)
    {
        if (s[i] == 'K')
        {
            countk++;
            table[j--, 1] = countk;
        }

        // Update maximum length
        if (Math.Min(table[j + 1, 0],
                     table[j + 1, 1]) > max)
            max = Math.Min(table[j + 1, 0],
                           table[j + 1, 1]);
    }

    return max;
}
```

```
// Driver code
static public void Main(String []srgs)
{
    String s = "RKRRRKKRRKKKKRR";
    int n = find(s);
    Console.WriteLine(n);
}
}

// This code is contributed
// by Arnab Kundu
```

Output:

```
4
```

Time Complexity: O(n) where ln is the length of the substring.

Improved By : andrew1234

Source

https://www.geeksforgeeks.org/maximum-length-subsequence-possible-of-the-form-rn-kn/

Chapter 200

Maximum length subsequence with difference between adjacent elements as either 0 or 1 | Set 2

Maximum length subsequence with difference between adjacent elements as either 0 or 1 | Set 2 - GeeksforGeeks

Given an array of **n** integers. The problem is to find maximum length of the subsequence with difference between adjacent elements in the subsequence as either 0 or 1. Time Complexity of O(n) is required.

Examples:

```
Input : arr[] = {2, 5, 6, 3, 7, 6, 5, 8}
Output : 5
The subsequence is {5, 6, 7, 6, 5}.

Input : arr[] = {-2, -1, 5, -1, 4, 0, 3}
Output : 4
The subsequence is {-2, -1, -1, 0}.
```

Method 1: Previously an approach having time complexity of $O(n^2)$ have been discussed in this post.

Method 2 (Efficient Approach): The idea is to create a hash map having tuples in the form **(ele, len)**, where **len** denotes the length of the longest subsequence ending with the element **ele**. Now, for each element arr[i] we can find the length of the values arr[i]-1, arr[i] and arr[i]+1 in the hash table and consider the maximum among them. Let this maximum

value be **max**. Now, the length of longest subsequence ending with arr[i] would be **max+1**. Update this length along with the element arr[i] in the hash table. Finally, the element having the maximum length in the hash table gives the maximum length subsequence.

```
 // C++ implementation to find maximum length
// subsequence with difference between adjacent
// elements as either 0 or 1
#include <bits/stdc++.h>
using namespace std;

// function to find maximum length subsequence
// with difference between adjacent elements as
// either 0 or 1
int maxLenSub(int arr[], int n)
{
    // hash table to map the array element with the
    // length of the longest subsequence of which
    // it is a part of and is the last element of
    // that subsequence
    unordered_map<int, int> um;

    // to store the maximum length subsequence
    int maxLen = 0;

    // traverse the array elements
    for (int i=0; i<n; i++)
    {
        // initialize current length
        // for element arr[i] as 0
        int len = 0;

        // if 'arr[i]-1' is in 'um' and its length of
        // subsequence is greater than 'len'
        if (um.find(arr[i]-1) != um.end() && len < um[arr[i]-1])
            len = um[arr[i]-1];

        // if 'arr[i]' is in 'um' and its length of
        // subsequence is greater than 'len'
        if (um.find(arr[i]) != um.end() && len < um[arr[i]])
            len = um[arr[i]];

        // if 'arr[i]+1' is in 'um' and its length of
        // subsequence is greater than 'len'
        if (um.find(arr[i]+1) != um.end() && len < um[arr[i]+1])
            len = um[arr[i]+1];

        // update arr[i] subsequence length in 'um'
        um[arr[i]] = len + 1;
```

```
        // update maximum length
        if (maxLen < um[arr[i]])
            maxLen = um[arr[i]];
    }

    // required maximum length subsequence
    return maxLen;
}

// Driver program to test above
int main()
{
    int arr[] = {2, 5, 6, 3, 7, 6, 5, 8};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum length subsequence = "
         << maxLenSub(arr, n);
    return 0;
}
```

Output:

```
Maximum length subsequence = 5
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Thanks to **Neeraj** for suggesting the above solution in the comments of this post.

Source

https://www.geeksforgeeks.org/maximum-length-subsequence-difference-adjacent-elements-either-0-1-set-2/

Chapter 201

Maximum number of characters between any two same character in a string

Maximum number of characters between any two same character in a string - GeeksforGeeks

Given a string, find the maximum number of characters between any two same character in the string. If no character repeats, print -1.

```
Input : str = "abba"
Output : 2
The maximum number of characters are
between two occurrences of 'a'.

Input : str = "baaabcddc"
Output : 3
The maximum number of characters are
between two occurrences of 'b'.

Input : str = "abc"
Output : -1
```

Approach 1 (Simple): Use two nested loops. The outer loop picks character from left to right, the inner loop finds farthest occurrence and keeps track of maximum.

C++

```
 // Simple C++ program to find maximum number
// of characters between two occurrences of
```

```
// same character
#include <bits/stdc++.h>
using namespace std;

int maximumChars(string& str)
{
    int n = str.length();
    int res = -1;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (str[i] == str[j])
                res = max(res, abs(j - i - 1));
    return res;
}

// Driver code
int main()
{
    string str = "abba";
    cout << maximumChars(str);
    return 0;
}
```

Java

```
 // Simple Java program to find maximum
// number of characters between two
// occurrences of same character
class GFG {

    static int maximumChars(String str)
    {
        int n = str.length();
        int res = -1;

        for (int i = 0; i < n - 1; i++)
            for (int j = i + 1; j < n; j++)
                if (str.charAt(i) == str.charAt(j))
                    res = Math.max(res,
                         Math.abs(j - i - 1));

        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        String str = "abba";
```

```
        System.out.println(maximumChars(str));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // Simple C# program to find maximum
// number of characters between two
// occurrences of same character
using System;

public class GFG {

    static int maximumChars(string str)
    {
        int n = str.Length;
        int res = -1;

        for (int i = 0; i < n - 1; i++)
            for (int j = i + 1; j < n; j++)
                if (str[i] == str[j])
                    res = Math.Max(res,
                         Math.Abs(j - i - 1));

        return res;
    }

    // Driver code
    static public void Main ()
    {
        string str = "abba";

        Console.WriteLine(maximumChars(str));
    }
}

// This code is contributed by vt_m.
```

Output:

```
2
```

Time Complexity : O(n^2)

Approach 2 (Efficient) : Initialize an array"FIRST" of length 26 in which we have to store first occurrence of an alphabet in the string and another array "LAST" of length 26 in which we will store last occurrence of the alphabet in the string here index 0 is correspond to alphabet a, 1 for b and so on . After that we will take the difference of last and first arrays to find max difference if they are not at same position.

C++

```
 // Efficient C++ program to find maximum number
// of characters between two occurrences of
// same character
#include <bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 256;

int maximumChars(string& str)
{
    int n = str.length();
    int res = -1;

    // Initialize all indexes as -1.
    int firstInd[MAX_CHAR];
    for (int i = 0; i < MAX_CHAR; i++)
        firstInd[i] = -1;

    for (int i = 0; i < n; i++) {
        int first_ind = firstInd[str[i]];

        // If this is first occurrence
        if (first_ind == -1)
            firstInd[str[i]] = i;

        // Else find distance from previous
        // occurrence and update result (if
        // required).
        else
            res = max(res, abs(i - first_ind - 1));
    }
    return res;
}

// Driver code
int main()
{
    string str = "abba";
    cout << maximumChars(str);
```

```
    return 0;
}
```

Java

```
 // Efficient java program to find maximum
// number of characters between two
// occurrences of same character
import java.io.*;

public class GFG {

    static int MAX_CHAR = 256;

    static int maximumChars(String str)
    {
        int n = str.length();
        int res = -1;

        // Initialize all indexes as -1.
        int []firstInd = new int[MAX_CHAR];

        for (int i = 0; i < MAX_CHAR; i++)
            firstInd[i] = -1;

        for (int i = 0; i < n; i++)
        {
            int first_ind = firstInd[str.charAt(i)];

            // If this is first occurrence
            if (first_ind == -1)
                firstInd[str.charAt(i)] = i;

            // Else find distance from previous
            // occurrence and update result (if
            // required).
            else
                res = Math.max(res, Math.abs(i
                                    - first_ind - 1));
        }

        return res;
    }

    // Driver code

    static public void main (String[] args)
    {
```

```
        String str = "abba";

        System.out.println(maximumChars(str));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // Efficient C# program to find maximum
// number of characters between two
// occurrences of same character
using System;

public class GFG {

    static int MAX_CHAR = 256;

    static int maximumChars(string str)
    {
        int n = str.Length;
        int res = -1;

        // Initialize all indexes as -1.
        int []firstInd = new int[MAX_CHAR];

        for (int i = 0; i < MAX_CHAR; i++)
            firstInd[i] = -1;

        for (int i = 0; i < n; i++)
        {
            int first_ind = firstInd[str[i]];

            // If this is first occurrence
            if (first_ind == -1)
                firstInd[str[i]] = i;

            // Else find distance from previous
            // occurrence and update result (if
            // required).
            else
                res = Math.Max(res, Math.Abs(i
                              - first_ind - 1));
        }

        return res;
    }
```

```
    // Driver code

    static public void Main ()
    {
        string str = "abba";

        Console.WriteLine(maximumChars(str));
    }
}

// This code is contributed by vt_m.
```

Output:

```
2
```

Time Complexity : O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-number-characters-two-character-string/

Chapter 202

Maximum number of chocolates to be distributed equally among k students

Maximum number of chocolates to be distributed equally among k students - GeeksforGeeks

Given **n** boxes containing some chocolates arranged in a row. There are **k** number of students. The problem is to distribute maximum number of chocolates equally among **k** students by selecting a consecutive sequence of boxes from the given lot. Consider the boxes are arranged in a row with numbers from 1 to n from left to right. We have to select a group of boxes which are in consecutive order that could provide maximum number of chocolates equally to all the **k** students. An array **arr[]** is given representing the row arrangement of the boxes and arr[i] represents number of chocolates in that box at position 'i'.

Examples:

```
Input : arr[] = {2, 7, 6, 1, 4, 5}, k = 3
Output : 6
The subarray is {7, 6, 1, 4} with sum 18.
Equal distribution of 18 chocolates among
3 students is 6.
Note that the selected boxes are in consecutive order
with indexes {1, 2, 3, 4}.
```

Source: Asked in Amazon.

The problem is to find maximum sum sub-array divisible by k and then return (sum / k).

Method 1 (Naive Approach): Consider the sum of all the sub-arrays. Select the maximum sum. Let it be **maxSum**. Return **(maxSum / k)**. Time Complexity is of $O(n^2)$.

Method 2 (Efficient Approach): Create an array **sum[]** where **sum[i]** stores **sum(arr[0]+..arr[i])**. Create a hash table having tuple as **(ele, idx)**, where ele represents

an element of **(sum[i] % k)** and **idx** represents the element's index of first occurrence when array **sum[]** is being traversed from left to right. Now traverse **sum[]** from i = 0 to n and follow the steps given below.

1. Calculate current remainder as **curr_rem** = sum[i] % k.
2. If curr_rem == 0, then check if maxSum < sum[i], update **maxSum** = sum[i].
3. Else if **curr_rem** is not present in the hash table, then create tuple **(curr_rem, i)** in the hash table.
4. Else, get the value associated with **curr_rem** in the hash table. Let this be **idx**. Now, if maxSum < (sum[i] – sum[idx]) then update **maxSum** = sum[i] – sum[idx].

Finally, return **(maxSum / k)**.

Explanation:
If (sum[i] % k) == (sum[j] % k), where sum[i] = sum(arr[0]+..+arr[i]) and sum[j] = sum(arr[0]+..+arr[j]) and 'i' is less than 'j', then sum(arr[i+1]+..+arr[j]) must be divisible by 'k'.

C++

```
 // C++ implementation to find the maximum number
// of chocolates to be distributed equally among
// k students
#include <bits/stdc++.h>
using namespace std;

// function to find the maximum number of chocolates
// to be distributed equally among k students
int maxNumOfChocolates(int arr[], int n, int k)
{
    // unordered_map 'um' implemented as
    // hash table
    unordered_map<int, int> um;

    // 'sum[]' to store cumulative sum, where
    // sum[i] = sum(arr[0]+..arr[i])
    int sum[n], curr_rem;

    // to store sum of sub-array having maximum sum
    int maxSum = 0;

    // building up 'sum[]'
    sum[0] = arr[0];
    for (int i = 1; i < n; i++)
        sum[i] = sum[i - 1] + arr[i];

    // traversing 'sum[]'
    for (int i = 0; i < n; i++) {
```

```
        // finding current remainder
        curr_rem = sum[i] % k;

        // if true then sum(0..i) is divisible
        // by k
        if (curr_rem == 0) {
            // update 'maxSum'
            if (maxSum < sum[i])
                maxSum = sum[i];
        }

        // if value 'curr_rem' not present in 'um'
        // then store it in 'um' with index of its
        // first occurrence
        else if (um.find(curr_rem) == um.end())
            um[curr_rem] = i;

        else
            // if true, then update 'max'
            if (maxSum < (sum[i] - sum[um[curr_rem]]))
            maxSum = sum[i] - sum[um[curr_rem]];
    }

    // required maximum number of chocolates to be
    // distributed equally among 'k' students
    return (maxSum / k);
}

// Driver program to test above
int main()
{
    int arr[] = { 2, 7, 6, 1, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << "Maximum number of chocolates: "
         << maxNumOfChocolates(arr, n, k);
    return 0;
}
```

Java

```
 // Java implementation to find the maximum number
// of chocolates to be distributed equally among
// k students
import java.io.*;
import java.util.*;
class GFG {
```

```
// Function to find the maximum number of chocolates
// to be distributed equally among k students
static int maxNumOfChocolates(int arr[], int n, int k)
{
    // Hash table
    HashMap <Integer,Integer> um = new HashMap<Integer,Integer>();

    // 'sum[]' to store cumulative sum, where
    // sum[i] = sum(arr[0]+..arr[i])
    int[] sum=new int[n];
    int curr_rem;

    // To store sum of sub-array having maximum sum
    int maxSum = 0;

    // Building up 'sum[]'
    sum[0] = arr[0];
    for (int i = 1; i < n; i++)
        sum[i] = sum[i - 1] + arr[i];

    // Traversing 'sum[]'
    for (int i = 0; i < n; i++) {

        // Finding current remainder
        curr_rem = sum[i] % k;

        // If true then sum(0..i) is divisible
        // by k
        if (curr_rem == 0) {
            // update 'maxSum'
            if (maxSum < sum[i])
                maxSum = sum[i];
        }

        // If value 'curr_rem' not present in 'um'
        // then store it in 'um' with index of its
        // first occurrence
        else if (!um.containsKey(curr_rem) )
            um.put(curr_rem , i);

        else
            // If true, then update 'max'
            if (maxSum < (sum[i] - sum[um.get(curr_rem)]))
            maxSum = sum[i] - sum[um.get(curr_rem)];
    }

    // Required maximum number of chocolates to be
    // distributed equally among 'k' students
```

```
    return (maxSum / k);
}

// Driver Code
public static void main(String[] args)
{
int arr[] = { 2, 7, 6, 1, 4, 5 };
int n = arr.length;
int k = 3;
System.out.println("Maximum number of chocolates: "
                    + maxNumOfChocolates(arr, n, k));
}
    }

// This code is contributed by 'Gitanjali'.
```

Python3

```
 # Python3 implementation to
# find the maximum number
# of chocolates to be
# distributed equally
# among k students

# function to find the
# maximum number of chocolates
# to be distributed equally
# among k students
def maxNumOfChocolates(arr, n, k):

    um, curr_rem, maxSum = {}, 0, 0

    # 'sm[]' to store cumulative sm,
    # where sm[i] = sm(arr[0]+..arr[i])
    sm = [0]*n
    sm[0] = arr[0]

    # building up 'sm[]'
    for i in range(1, n):
        sm[i] = sm[i - 1] + arr[i]

    # traversing 'sm[]'
    for i in range(n):

        # finding current remainder
        curr_rem = sm[i] % k

        if (not curr_rem and maxSum < sm[i]) :
```

```
            maxSum = sm[i]
        elif (not curr_rem in um) :
            um[curr_rem] = i
        elif (maxSum < (sm[i] - sm[um[curr_rem]])):
              maxSum = sm[i] - sm[um[curr_rem]]

    return maxSum//k

# Driver program to test above
arr = [ 2, 7, 6, 1, 4, 5 ]
n, k = len(arr), 3

print("Maximum number of chocolates: " +
     str(maxNumOfChocolates(arr, n, k)))

# This code is contributed by Ansu Kumari
```

C#

```
 // C# implementation to find
// the maximum number of
// chocolates to be distributed
// equally among k students
using System;
using System.Collections.Generic;

class GFG
{
    // Function to find the
    // maximum number of
    // chocolates to be distributed
    // equally among k students
    static int maxNumOfChocolates(int []arr,
                                  int n, int k)
    {
        // Hash table
        Dictionary <int, int> um =
                    new Dictionary<int, int>();

        // 'sum[]' to store cumulative
        // sum, where sum[i] =
        // sum(arr[0]+..arr[i])
        int[] sum = new int[n];
        int curr_rem;

        // To store sum of sub-array
        // having maximum sum
        int maxSum = 0;
```

```
    // Building up 'sum[]'
    sum[0] = arr[0];
    for (int i = 1; i < n; i++)
        sum[i] = sum[i - 1] + arr[i];

    // Traversing 'sum[]'
    for (int i = 0; i < n; i++)
    {

        // Finding current
        // remainder
        curr_rem = sum[i] % k;

        // If true then sum(0..i)
        // is divisible by k
        if (curr_rem == 0)
        {
            // update 'maxSum'
            if (maxSum < sum[i])
                maxSum = sum[i];
        }

        // If value 'curr_rem' not
        // present in 'um' then store
        // it in 'um' with index of
        // its first occurrence
        else if (!um.ContainsKey(curr_rem))
            um.Add(curr_rem , i);

        else

            // If true, then
            // update 'max'
            if (maxSum < (sum[i] -
                sum[um[curr_rem]]))
            maxSum = sum[i] -
                     sum[um[curr_rem]];
    }

    // Required maximum number
    // of chocolates to be
    // distributed equally
    // among 'k' students
    return (maxSum / k);
}

// Driver Code
```

```
    static void Main()
    {
    int []arr = new int[]{ 2, 7, 6, 1, 4, 5 };
    int n = arr.Length;
    int k = 3;
    Console.Write("Maximum number of chocolates: " +
                    maxNumOfChocolates(arr, n, k));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
Maximum number of chocolates: 6
```

Time Complexity: O(n).
Auxiliary Space: O(n).

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/maximum-number-chocolates-distributed-equally-among-k-students/

Chapter 203

Maximum occurring character in an input string | Set-2

Maximum occurring character in an input string | Set-2 - GeeksforGeeks

Given a string containing lowercase characters. The task is to print the maximum occurring character in the input string. If 2 or more characters appear the same number of times, print the lexicographically (alphabetically) lowest (first) character.

Examples:

> **Input:** test sample
> **Output:** e
> 't', 'e' and 's' appears 2 times, but 'e' is the lexicographically smallest character.
>
> **Input:** sample program
> **Output:** a

In the previous article, if there are more than one characters occurring the maximum number of time, then any of the characters is returned. In this post, the lexicographically smallest character of all the characters is returned.

Approach: Declare a **freq[26]** array which is used as a hash table to store the frequencies of each character in the input string. Iterate in the string and increase the count of *freq[s[i]]* for every character.T raverse the **freq[]** array from left to right and keep track of the character having the maximum frequency so far. The value at freq[i] represents the frequency of character (i + 'a').

Below is the implementation of the above approach:

```
 // C++ implementation to find
// the maximum occurring character in
// an input string which is lexicographically first
#include <bits/stdc++.h>
```

```
using namespace std;

// function to find the maximum occurring character in
// an input string which is lexicographically first
char getMaxOccurringChar(char str[])
{
    // freq[] used as hash table
    int freq[26] = { 0 };

    // to store maximum frequency
    int max = -1;

    // to store the maximum occurring character
    char result;

    // length of 'str'
    int len = strlen(str);

    // get frequency of each character of 'str'
    for (int i = 0; i < len; i++)
        freq[str[i] - 'a']++;

    // for each character, where character is obtained by
    // (i + 'a') check whether it is the maximum character
    // so far and accodingly update 'result'
    for (int i = 0; i < 26; i++)
        if (max < freq[i]) {
            max = freq[i];
            result = (char)(i + 'a');
        }

    // maximum occurring character
    return result;
}

// Driver Code
int main()
{
    char str[] = "sample program";
    cout << "Maximum occurring character = "
         << getMaxOccurringChar(str);
    return 0;
}
```

Output:

```
Maximum occurring character = a
```

Time Complexity: O(n).
Auxiliary Space: O(1).

Source: Sabre Interview Experience | Set 2

Source

https://www.geeksforgeeks.org/maximum-occurring-character-in-an-input-string-set-2/

Chapter 204

Maximum possible difference of two subsets of an array

Maximum possible difference of two subsets of an array - GeeksforGeeks

Given an array of n-integers. Array may contain repetitive elements but the highest frequency of any elements must not exceed two. You have to make two subsets such that difference of their elements sum is maximum and both of them jointly contains all of elements of given array along with the most important condition, no subset should contain repetitive elements.

Examples:

```
Input : arr[] = {5, 8, -1, 4}
Output : Maximum Difference = 18
Explanation :
Let Subset A = {5, 8, 4} & Subset B = {-1}
Sum of elements of subset A = 17, of subset B = -1
Difference of Sum of Both subsets = 17 - (-1) = 18

Input : arr[] = {5, 8, 5, 4}
Output : Maximum Difference = 12
Explanation :
Let Subset A = {5, 8, 4} & Subset B = {5}
Sum of elements of subset A = 17, of subset B = 5
Difference of Sum of Both subsets = 17 - 5 = 12
```

Before solving this question we have to take care of some given conditions and they are listed as:

- While building up the subsets, take care that no subset should contain repetitive elements. And for this we can conclude that all such elements whose frequency are

2, going to be part of both subsets and hence overall they don't have any impact on difference of subset sum. So, we can easily ignore them.
- For making the difference of sum of elements of both subset maximum we have to make subset in such a way that all positive elements belongs to one subset and negative ones to other subset.

Algorithm with time complexity $O(n^2)$:

```
for i=0 to n-1
    isSingleOccurance = true;
    for  j= i+1 to n-1

        // if frequency of any element is two
        // make both equal to zero
        if arr[i] equals arr[j]
            arr[i] = arr[j] = 0
            isSingleOccurance = false;
            break;

    if isSingleOccurance == true
        if (arr[i] > 0)
            SubsetSum_1 += arr[i];
        else
            SubsetSum_2 += arr[i];
return abs(SubsetSum_1 - SubsetSum2)
```

C++

```
 // CPP find maximum difference of subset sum
#include <bits/stdc++.h>
using namespace std;

// function for maximum subset diff
int maxDiff(int arr[], int n)
{
    int SubsetSum_1 = 0, SubsetSum_2 = 0;
    for (int i = 0; i <= n - 1; i++) {

        bool isSingleOccurance = true;
        for (int j = i + 1; j <= n - 1; j++) {

            // if frequency of any element is two
            // make both equal to zero
            if (arr[i] == arr[j]) {
                isSingleOccurance = false;
                arr[i] = arr[j] = 0;
```

```
                break;
            }
        }
        if (isSingleOccurance) {
            if (arr[i] > 0)
                SubsetSum_1 += arr[i];
            else
                SubsetSum_2 += arr[i];
        }
    }
    return abs(SubsetSum_1 - SubsetSum_2);
}

// driver program
int main()
{
    int arr[] = { 4, 2, -3, 3, -2, -2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = " << maxDiff(arr, n);
    return 0;
}
```

Java

```
 // java find maximum difference
// of subset sum
import java .io.*;

public class GFG {

    // function for maximum subset diff
    static int maxDiff(int []arr, int n)
    {
        int SubsetSum_1 = 0, SubsetSum_2 = 0;
        for (int i = 0; i <= n - 1; i++)
        {
            boolean isSingleOccurance = true;
            for (int j = i + 1; j <= n - 1; j++)
            {

                // if frequency of any element
                // is two make both equal to
                // zero
                if (arr[i] == arr[j])
                {
                    isSingleOccurance = false;
                    arr[i] = arr[j] = 0;
                    break;
```

```
                }
            }
            if (isSingleOccurance)
            {
                if (arr[i] > 0)
                    SubsetSum_1 += arr[i];
                else
                    SubsetSum_2 += arr[i];
            }
        }

        return Math.abs(SubsetSum_1 - SubsetSum_2);
    }

    // driver program
    static public void main (String[] args)
    {
        int []arr = { 4, 2, -3, 3, -2, -2, 8 };
        int n = arr.length;

        System.out.println("Maximum Difference = "
                                + maxDiff(arr, n));
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Python3 find maximum difference
# of subset sum

import math

# function for maximum subset diff
def maxDiff(arr, n) :
    SubsetSum_1 = 0
    SubsetSum_2 = 0
    for i in range(0, n) :

        isSingleOccurance = True
        for j in range(i + 1, n) :

            # if frequency of any element
            # is two make both equal to
            # zero
            if (arr[i] == arr[j]) :
                isSingleOccurance = False
```

```
                arr[i] = arr[j] = 0
                break

        if (isSingleOccurance == True) :
            if (arr[i] > 0) :
                SubsetSum_1 += arr[i]
            else :
                SubsetSum_2 += arr[i]

    return abs(SubsetSum_1 - SubsetSum_2)

# Driver Code
arr = [4, 2, -3, 3, -2, -2, 8]
n = len(arr)
print ("Maximum Difference = {}"
               . format(maxDiff(arr, n)))

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
 // C# find maximum difference of
// subset sum
using System;

public class GFG {

    // function for maximum subset diff
    static int maxDiff(int []arr, int n)
    {
        int SubsetSum_1 = 0, SubsetSum_2 = 0;
        for (int i = 0; i <= n - 1; i++)
        {

            bool isSingleOccurance = true;
            for (int j = i + 1; j <= n - 1; j++)
            {

                // if frequency of any element
                // is two make both equal to
                // zero
                if (arr[i] == arr[j])
                {
                    isSingleOccurance = false;
                    arr[i] = arr[j] = 0;
                    break;
                }
```

```
            }
            if (isSingleOccurance)
            {
                if (arr[i] > 0)
                    SubsetSum_1 += arr[i];
                else
                    SubsetSum_2 += arr[i];
            }
        }

        return Math.Abs(SubsetSum_1 - SubsetSum_2);
    }

    // driver program
    static public void Main ()
    {
        int []arr = { 4, 2, -3, 3, -2, -2, 8 };
        int n = arr.Length;

        Console.WriteLine("Maximum Difference = "
                              + maxDiff(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP find maximum difference
// of subset sum

// function for maximum subset diff
function maxDiff($arr, $n)
{
    $SubsetSum_1 = 0;
    $SubsetSum_2 = 0;
    for ($i = 0; $i <= $n - 1; $i++)
    {

        $isSingleOccurance = true;
        for ($j = $i + 1; $j <= $n - 1; $j++)
        {

            // if frequency of any element is two
            // make both equal to zero
            if ($arr[$i] == $arr[$j])
            {
```

```
                $isSingleOccurance = false;
                $arr[$i] = $arr[$j] = 0;
                break;
            }
        }
        if ($isSingleOccurance)
        {
            if ($arr[$i] > 0)
                $SubsetSum_1 += $arr[$i];
            else
                $SubsetSum_2 += $arr[$i];
        }
    }
    return abs($SubsetSum_1 - $SubsetSum_2);
}

    // Driver Code
    $arr = array(4, 2, -3, 3, -2, -2, 8);
    $n = sizeof($arr);
    echo "Maximum Difference = " , maxDiff($arr, $n);

// This code is contributed by nitin mittal
?>
```

Output:

```
Maximum Difference = 20
```

Algorithm with time complexity O(n log n):

```
-> sort the array
-> for i =0 to n-2
      // consecutive two elements are not equal
      // add absolute arr[i] to result
      if arr[i] != arr[i+1]
          result += abs(arr[i])
      // else skip next element too
      else
          i++;

// special check for last two elements
-> if (arr[n-2] != arr[n-1])
    result += arr[n-1]

-> return result;
```

C++

```
 // CPP find maximum difference of subset sum
#include <bits/stdc++.h>
using namespace std;

// function for maximum subset diff
int maxDiff(int arr[], int n)
{
    int result = 0;

    // sort the array
    sort(arr, arr + n);

    // calculate the result
    for (int i = 0; i < n - 1; i++) {
        if (arr[i] != arr[i + 1])
            result += abs(arr[i]);
        else
            i++;
    }

    // check for last element
    if (arr[n - 2] != arr[n - 1])
        result += abs(arr[n - 1]);

    // return result
    return result;
}

// driver program
int main()
{
    int arr[] = { 4, 2, -3, 3, -2, -2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = " << maxDiff(arr, n);
    return 0;
}
```

Java

```
 // java find maximum difference of
// subset sum
import java. io.*;
import java .util.*;

public class GFG {
```

```
    // function for maximum subset diff
    static int maxDiff(int []arr, int n)
    {
        int result = 0;

        // sort the array
        Arrays.sort(arr);

        // calculate the result
        for (int i = 0; i < n - 1; i++)
        {
            if (arr[i] != arr[i + 1])
                result += Math.abs(arr[i]);
            else
                i++;
        }

        // check for last element
        if (arr[n - 2] != arr[n - 1])
            result += Math.abs(arr[n - 1]);

        // return result
        return result;
    }

    // driver program
    static public void main (String[] args)
    {
        int[] arr = { 4, 2, -3, 3, -2, -2, 8 };
        int n = arr.length;

        System.out.println("Maximum Difference = "
                                + maxDiff(arr, n));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# find maximum difference
// of subset sum
using System;

public class GFG {

    // function for maximum subset diff
```

```
    static int maxDiff(int []arr, int n)
    {
        int result = 0;

        // sort the array
        Array.Sort(arr);

        // calculate the result
        for (int i = 0; i < n - 1; i++)
        {
            if (arr[i] != arr[i + 1])
                result += Math.Abs(arr[i]);
            else
                i++;
        }

        // check for last element
        if (arr[n - 2] != arr[n - 1])
            result += Math.Abs(arr[n - 1]);

        // return result
        return result;
    }

    // driver program
    static public void Main ()
    {
        int[] arr = { 4, 2, -3, 3, -2, -2, 8 };
        int n = arr.Length;

        Console.WriteLine("Maximum Difference = "
                              + maxDiff(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP find maximum difference of subset sum

// function for maximum subset diff
function maxDiff( $arr, $n)
{
    $result = 0;

    // sort the array
```

```
    sort($arr);

    // calculate the result
    for ( $i = 0; $i < $n - 1; $i++)
    {
        if ($arr[$i] != $arr[$i + 1])
            $result += abs($arr[$i]);
        else
            $i++;
    }

    // check for last element
    if ($arr[$n - 2] != $arr[$n - 1])
        $result += abs($arr[$n - 1]);

    // return result
    return $result;
}

    // Driver Code
    $arr = array( 4, 2, -3, 3, -2, -2, 8 );
    $n = count($arr);
    echo "Maximum Difference = "
        , maxDiff($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Maximum Difference = 20
```

Algorithm with time complexity O(n):

```
make hash table for positive elements:
    for all positive elements(arr[i])
        if frequency == 1
            SubsetSum_1 += arr[i];
make hash table for negative elements:
    for all negative elements
        if frequency == 1
            SubsetSum_2 += arr[i];
return abs(SubsetSum_1 - SubsetSum2)

 // CPP find maximum difference of subset sum
```

```
#include <bits/stdc++.h>
using namespace std;

// function for maximum subset diff
int maxDiff(int arr[], int n)
{
    unordered_map<int, int> hashPositive;
    unordered_map<int, int> hashNegative;

    int SubsetSum_1 = 0, SubsetSum_2 = 0;

    // construct hash for positive elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] > 0)
            hashPositive[arr[i]]++;

    // calculate subset sum for positive elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] > 0 && hashPositive[arr[i]] == 1)
            SubsetSum_1 += arr[i];

    // construct hash for negative elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] < 0)
            hashNegative[abs(arr[i])]++;

    // calculate subset sum for negative elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] < 0 &&
            hashNegative[abs(arr[i])] == 1)
            SubsetSum_2 += arr[i];

    return abs(SubsetSum_1 - SubsetSum_2);
}

// driver program
int main()
{
    int arr[] = { 4, 2, -3, 3, -2, -2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = " << maxDiff(arr, n);
    return 0;
}
```

Output:

```
Maximum Difference = 20
```

Improved By : vt_m, nitin mittal, manishshaw1

Source

https://www.geeksforgeeks.org/maximum-possible-difference-two-subsets-array/

Chapter 205

Maximum possible sum of a window in an array such that elements of same window in other array are unique

Maximum possible sum of a window in an array such that elements of same window in other array are unique - GeeksforGeeks

Given two arrays A and B of equal number of elements. Task is to find the maximum sum possible of a window in array B such that elements of same window in A[] are unique.

Examples:

```
Input : A = [0, 1, 2, 3, 0, 1, 4]
        B = [9, 8, 1, 2, 3, 4, 5]
Output : sum = 20
The maximum sum possible in B[] such that
all corresponding elements in A[] are unique
is (9+8+1+2) = 20.

Input : A = [0, 1, 2, 0, 2]
        B = [5, 6, 7, 8, 2]
Output :sum = 21
```

A **simple solution** is to consider all subarrays of B[]. For every subarray, check if elements same subarray in A[] are distinct or not. If distinct, then compare sum with result and update result.
Time complexity of this solution is $O(n^2)$

An **efficient solution** is to use hashing.

1. Create an empty hash table.
2. Traverse array elements. Do following for every element A[i].
    - While A[i] is present in hash table, keep removing elements from beginning of current window and keep subtracting window beginning element of B[] from current sum.
3. Add B[i] to current sum and update result if current sum becomes more.
4. Return result.

Below is C++ implementation of above steps.

C++

```
 // C++ program to find the maximum
// possible sum of a window in one
// array such that elements in same
// window of other array are unique.
#include <bits/stdc++.h>
using namespace std;

// Function to return maximum sum of window
// in B[] according to given constraints.
int returnMaxSum(int A[], int B[], int n)
{
    // Map is used to store elements
    // and their counts.
    unordered_set<int> mp;

    int result = 0; // Initialize result

    // calculating the maximum possible
    // sum for each subarray containing
    // unique elements.
    int curr_sum = 0, curr_begin = 0;
    for (int i = 0; i < n; ++i) {

        // Remove all duplicate
        // instances of A[i] in
        // current window.
        while (mp.find(A[i]) != mp.end()) {
            mp.erase(A[curr_begin]);
            curr_sum -= B[curr_begin];
            curr_begin++;
        }

        // Add current instance of A[i]
        // to map and to current sum.
```

```
        mp.insert(A[i]);
        curr_sum += B[i];

        // Update result if current
        // sum is more.
        result = max(result, curr_sum);
    }

    return result;
}

// Driver code
int main()
{
    int A[] = { 0, 1, 2, 3, 0, 1, 4 };
    int B[] = { 9, 8, 1, 2, 3, 4, 5 };
    int n = sizeof(A)/sizeof(A[0]);
    cout << returnMaxSum(A, B, n);
    return 0;
}
```

Java

```
 // Java program to find the maximum
// possible sum of a window in one
// array such that elements in same
// window of other array are unique.
import java.util.HashSet;
import java.util.Set;

public class MaxPossibleSuminWindow
{
    // Function to return maximum sum of window
    // in A[] according to given constraints.
    static int returnMaxSum(int A[], int B[], int n)
    {

        // Map is used to store elements
        // and their counts.
        Set<Integer> mp = new HashSet<Integer>();

        int result = 0; // Initialize result

        // calculating the maximum possible
        // sum for each subarray containing
        // unique elements.
        int curr_sum = 0, curr_begin = 0;
        for (int i = 0; i < n; ++i)
```

```
        {
            // Remove all duplicate
            // instances of A[i] in
            // current window.
            while (mp.contains(A[i]))
            {
                mp.remove(A[curr_begin]);
                curr_sum -= B[curr_begin];
                curr_begin++;
            }

            // Add current instance of A[i]
            // to map and to current sum.
            mp.add(A[i]);
            curr_sum += B[i];

            // Update result if current
            // sum is more.
            result = Integer.max(result, curr_sum);

        }
        return result;
    }

    //Driver Code to test above method
    public static void main(String[] args)
    {
        int A[] = { 0, 1, 2, 3, 0, 1, 4 };
        int B[] = { 9, 8, 1, 2, 3, 4, 5 };
        int n = A.length;
        System.out.println(returnMaxSum(A, B, n));
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
20
```

Time complexity of this solution is O(n). Note that every element of array is inserted and removed at most once from array.

Source

https://www.geeksforgeeks.org/maximum-possible-sum-window-array-elements-window-array-unique/

Chapter 206

Minimum Index Sum for Common Elements of Two Lists

Minimum Index Sum for Common Elements of Two Lists - GeeksforGeeks

Ram and Shyam want to choose a website to learn programming and they both have a list of favorite websites represented by strings.
You need to help them find out their common interest with the least index sum. If there is a choice tie between answers, print all of them with no order requirement. Assume there always exists an answer.

Examples:

```
Input : ["GeeksforGeeks", "Udemy", "Coursera", "edX"]
        ["Codecademy", "Khan Academy", "GeeksforGeeks"]
Output : "GeeksforGeeks"
Explanation : GeeksforGeeks is the only common website
              in two lists

Input : ["Udemy", "GeeksforGeeks", "Coursera", "edX"]
        ["GeeksforGeeks", "Udemy", "Khan Academy", "Udacity"]
Output : "GeeksforGeeks" "Udemy"
Explanation : There are two common websites and index sum
              of both is same.
```

Naive Method:

The idea is to try all index sums from 0 to sum of sizes. For every sum, check if there are pairs with given sum. Once we find one or more pairs, we print them and return.

```
 #include <bits/stdc++.h>
using namespace std;
```

```
// Function to print common strings with minimum index sum
void find(vector<string> list1, vector<string> list2)
{
    vector<string> res; // resultant list
    int max_possible_sum = list1.size() + list2.size() - 2;

    // iterating over sum in ascending order
    for (int sum = 0; sum <= max_possible_sum ; sum++)
    {
        // iterating over one list and check index
        // (Corresponding to given sum) in other list
        for (int i = 0; i <= sum; i++)

            // put common strings in resultant list
            if (i < list1.size() &&
                (sum - i) < list2.size() &&
                list1[i] == list2[sum - i])
                res.push_back(list1[i]);

        // if common string found then break as we are
        // considering index sums in increasing order.
        if (res.size() > 0)
            break;
    }

    // print the resultant list
    for (int i = 0; i < res.size(); i++)
        cout << res[i] << " ";
}

// Driver code
int main()
{
    // Creating list1
    vector<string> list1;
    list1.push_back("GeeksforGeeks");
    list1.push_back("Udemy");
    list1.push_back("Coursera");
    list1.push_back("edX");

    // Creating list2
    vector<string> list2;
    list2.push_back("Codecademy");
    list2.push_back("Khan Academy");
    list2.push_back("GeeksforGeeks");

    find(list1, list2);
```

```
    return 0;
}
```

Output:

```
GeeksforGeeks
```

Time Complexity : O($(l_1+l_2)^2$ *x), where l_1 and l_2 are the lengths of list1 and list2 respectively and x refers to string length.

Auxiliary Space : O(l*x), where x refers to length of resultant list and l is length of maximum size word.

Using Hash:

1. Traverse over the list1 and create an entry for index each element of list1 in a Hash Table.
2. Traverse over list2 and for every element, check if the same element already exists as a key in the map. If so, it means that the element exists in both the lists.
3. Find out the sum of indices corresponding to common element in the two lists. If this sum is lesser than the minimum sum obtained till now, update the resultant list.
4. If the sum is equal to the minimum sum obtained till now, put an extra entry corresponding to the element in list2 in the resultant list.

Source

https://www.geeksforgeeks.org/minimum-index-sum-common-elements-two-lists/

Chapter 207

Minimum array element changes to make its elements 1 to N

Minimum array element changes to make its elements 1 to N - GeeksforGeeks

Suppose you are given an array with N elements with any integer values. You need to find minimum number of elements of the array which must be changed so that array has all integer values between 1 and N(including 1, N).

Examples:

```
Input : arr[] = {1 4 5 3 7}
Output : 1
We need to replace 7 with 2 to satisfy
condition hence minimum changes is 1.

Input : arr[] = {8 55 22 1 3 22 4 5}
Output :3
```

We insert all elements in a hash table. We then iterate from 1 to N and check whether the element in present in hash table. If it is not present than increment count. Final value of count will be the minimum changes required.

C++

```
 // Count minimum changes to make array
// from 1 to n
#include <bits/stdc++.h>
using namespace std;

int countChanges(int arr[], int n)
{
```

```
    // it will contain all initial elements
    // of array for log(n) complexity searching
    unordered_set<int> s;

    // Inserting all elements in a hash table
    for (int i = 0; i < n; i++)
        s.insert(arr[i]);

    // Finding elements to be changed
    int count = 0;
    for (int i = 1; i <= n; i++)
        if (s.find(i) == s.end())
            count++;

    return count;
}

int main()
{
    int arr[] = {8, 55, 22, 1, 3, 22, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countChanges(arr, n);
    return 0;
}
```

Python 3

```
# Count minimum changes to
# make array from 1 to n

def countChanges(arr, n):

# it will contain all initial
# elements of array for log(n)
# complexity searching
s = []

# Inserting all elements in a list
for i in range(n):
s.append(arr[i])

# Finding elements to be changed
count = 0
for i in range(1, n + 1) :
if i not in s:
count += 1

return count

# Driver Code
if ___name___ == "___main___":
```

```
arr = [8, 55, 22, 1, 3, 22, 4, 5]
n = len(arr)
print(countChanges(arr, n))

# This code is contributed
# by ChitraNayal
```

PHP

Output:

```
3
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Improved By : ChitraNayal

Source

https://www.geeksforgeeks.org/minimum-array-element-changes-to-make-its-elements-1-to-n/

Chapter 208

Minimum cost to construct a string

Minimum cost to construct a string - GeeksforGeeks

Given a string s (containing lowercase letters only), we have to find the minimum cost to construct the given string. The cost can determined using the following operations:
1. Appending a single character cost 1 unit
2. A sub-string of new string(intermediate string) can be appended without any cost

Note* Intermediate string is the string formed so far.
Examples:

```
Input : "geks"
Output : cost: 4
Explanation:
appending 'g' cost 1, string "g"
appending 'e' cost 1, string "ge"
appending 'k' cost 1, string "gek"
appending 's' cost 1, string "geks"
Hence, Total cost to construct "geks" is 4

Input :  "abab"
Output : cost: 2
Explanation:
Appending 'a' cost 1, string "a"
Appending 'b' cost 1, string "ab"
Appending  "ab" cost nothing as it
is substring of intermediate.
Hence, Total cost to construct "abab" is 2
```

Naive Approach: Check if there is sub-string in the remaining string to be constructed

which is also a sub-string in the intermediate string, if there is then append it at no cost and if not then append it at the cost of 1 unit per character.

In the above example when intermediate string was “ab” and we need to construct “abab” then remaining string was “ab”. Hence there is a sub-string in remaining string which is also a sub-string of intermediate string (i.e. “ab”) and therefore cost us nothing.

Better Approach: We will use hashing technique, to maintain that whether we have seen a character or not. If we have seen the character, then there is no cost to append the character and if not, then it cost us 1 unit.

Now in this approach we take one character at a time and not a string. This is because if “ab” is substring of “abab”, so is ‘a’ and ‘b’ alone and hence make no difference.

This also leads us to the conclusion that the cost to construct a string is never more than 26 in case the string contains all the alphabets (a-z).

```
 // C++ Program to find minimum cost to
// construct a string
#include <iostream>
using namespace std;

int minCost(string& s)
{
    // Initially all characters are un-seen
    bool alphabets[26] = { false };

    // Marking seen characters
    for (int i = 0; i < s.size(); i++)
        alphabets[s[i] - 97] = true;

    // Count total seen character, and that
    // is the cost
    int count = 0;
    for (int i = 0; i < 26; i++)
        if (alphabets[i])
            count++;

    return count;
}

int main()
{
    // s is the string that needs to be constructed
    string s = "geeksforgeeks";

    cout << "Total cost to construct "
         << s << " is " << minCost;

    return 0;
```

```
}
```

Output:

```
Total cost to construct geeksforgeeks is 7
```

Source

https://www.geeksforgeeks.org/minimum-cost-construct-string/

Chapter 209

Minimum delete operations to make all elements of array same

Minimum delete operations to make all elements of array same - GeeksforGeeks

Given an array of n elements such that elements may repeat. We can delete any number of elements from array. The task is to find minimum number of elements to be deleted from array to make it equal.

Examples:

```
Input  : arr[] = {4, 3, 4, 4, 2, 4}
Output : 2
After deleting 2 and 3 from array, array becomes
arr[] = {4, 4, 4, 4}

Input : arr[] = {1, 2, 3, 4, 5}
Output: 4
We can delete any four elements from array.
```

In this problem we need to minimize the delete operations. The approach is simple, we count frequency of each element in array, then find the frequency of most frequent element in **count array**. Let this frequence be max_freq. To get the minimum number of elements to be deleted from array calculate **n − max_freq** where n is number of elements in given array.

```
 // C++ program to find minimum number of deletes required
// to make all elements same.
#include<bits/stdc++.h>
using namespace std;
```

```
// Function to get minimum number of elements to be deleted
// from array to make array elements equal
int minDelete(int arr[],int n)
{
    // Create an hash map and store frequencies of all
    // array elements in it using element as key and
    // frequency as value
    unordered_map<int, int> freq;
    for (int i=0; i<n; i++)
        freq[arr[i]]++;

    // Find maximum frequency among all frequencies.
    int max_freq = INT_MIN;
    for (auto itr = freq.begin(); itr != freq.end(); itr++)
        max_freq = max(max_freq, itr->second);

    // To minimize delete operations, we remove all
    // elements but the most frequent element.
    return n - max_freq;
}

// Driver program to run the case
int main()
{
    int arr[] = {4, 3, 4, 4, 2, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << minDelete(arr, n);
    return 0;
}
```

Output:

```
2
```

Time complexity : O(n)

Note : Here we can optimize the extra space to count frequency of each element to O(1) but for this we have to modify our original array. See this article.

Source

https://www.geeksforgeeks.org/minimum-delete-operations-make-elements-array/

Chapter 210

Minimum equal palindromic cuts with rearrangements allowed

Minimum equal palindromic cuts with rearrangements allowed - GeeksforGeeks

Given a string of length n. Find the minimum number of possible cuts after rearranging the string (if required), such that each cut is a palindrome and length of every cut is equal. That is, find the minimum number of palindromes of equal lengths that can be obtained by partitioning the given string if rearrangement of string is allowed before partitioning.

Examples:

```
Input : string = "aabaac"
Output : 2
Explanation : Rearrange the string as "abaaca"
              and cut into "aba" and "aca"

Input : string = "aabbccdd"
Output : 1
Explanation : Rearrange the string as "abcddcba"
              This is a palindrome and cannot be
              cut further.
```

If we observe carefully, our problem reduces to calculating characters with odds and even counts. Below are the possible cases,

1. If the characters present in the string have only even counts then the answer will be 1 as we can rearrange the entire string to form a palindrome.
2. If there is only one character with odd count, then also the answer will be 1 as we can rearrange the entire string to form a palindrome.

3. If there is more than one character with odd count, then we will create two separate list of characters – one for odd characters and one for even characters. Now, if we notice that if a character has odd count then if we subtract 1 from it, the count will become even. So we will insert the element with odd counts only once in the odd list. We will insert the elements with even counts (evenCount/2) times, i.e. half of their count in the even list. Now our problem is to uniformly distribute the even count elements among odd count elements to form palindromes of equal length. Suppose the list of even count characters is *even* and odd count characters is *odd*. If even.size() is divisible by odd.size() our answer will be odd.size() otherwise we will transfer elements from even list to odd list until even.size() is divisible by odd.size().

Below is the implementation of above idea:

```
 // CPP program to find minimum number of palindromic
// cuts of equal length
#include<bits/stdc++.h>
using namespace std;

// function to find minimum number of
// palindromic cuts of equal length
int minPalindromeCuts(string str)
{
    // map to store count of characters
    unordered_map<char,int> m;

    // store count of characters in a map
    for (int i=0;i<str.length();i++)
    {
       if (m.find(str[i])==m.end())
            m.insert(make_pair(str[i],1));
       else
            m[str[i]]++;
    }

    // list to store even count characters
    vector<char> even;

    // list to store odd count characters
    vector<char> odd;

    for (auto itr = m.begin(); itr!=m.end(); itr++)
    {
        // add odd count characters only once and
        // decrement count by 1
        if (itr->second%2!=0)
        {
            odd.push_back(itr->first);
            itr->second--;
```

```
        }
    }

    for (auto itr = m.begin(); itr!=m.end(); itr++)
    {
        if (itr->second%2==0)
        {
            // add even count characters half of their
            // count to the even list so that we can
            // simply repeat the even list on both
            // sides of an odd char to generate a
            // palindrome
            for (int i=0;i<(itr->second)/2;i++)
                even.push_back(itr->first);
        }
    }

    // if there is no odd count characters or
    // only 1 odd count character, return 1
    if (odd.size() <= 1)
        return 1;

    else
    {
        // Move some characters from even list over
        // to odd list to make palindrome work
        while (odd.size() > 0 && even.size() > 0 &&
               even.size() % odd.size() != 0)
        {
            odd.push_back(even.back());
            odd.push_back(even.back());
            even.pop_back();
        }

        return odd.size();
    }
}

// driver code
int main()
{
    string str = "aabaac";
    cout << minPalindromeCuts(str);
    return 0;
}
```

Output:

2

Source

https://www.geeksforgeeks.org/minimum-equal-palindromic-cuts-with-rearrangements-allowed/

Chapter 211

Minimum insertions to form a palindrome with permutations allowed

Minimum insertions to form a palindrome with permutations allowed - GeeksforGeeks

Given a string of lowercase letters. Find minimum characters to be inserted in string so that it can become palindrome. We can change positions of characters in string.

Examples:

```
Input : geeksforgeeks
Output : 2
geeksforgeeks can be changed as:
geeksroforskeeg
geeksorfroskeeg
and many more

Input : aabbc
Output : 0
aabbc can be changed as:
abcba
bacab
```

A palindromic string can have one odd character only when length of string is odd otherwise all characters occur even number of times. So, we have to find characters which occur odd times in a string.

The idea is to count occurrence of each character in a string. As palindromic string can have one character which occur odd times so number of insertion will be one less then count

of characters which occur odd times. And if string is already palindrome, we do not need to add any character so result will be 0.

C++

```
 // CPP program to find minimum number
// of insertions to make a string
// palindrome
#include <bits/stdc++.h>
using namespace std;

// Function will return number of
// characters to be added
int minInsertion(string str)
{
    // To store string length
    int n = str.length();

    // To store number of characters
    // occurring odd number of times
    int res = 0;

    // To store count of each
    // character
    int count[26] = { 0 };

    // To store occurrence of each
    // character
    for (int i = 0; i < n; i++)
        count[str[i] - 'a']++;

    // To count characters with odd
    // occurrence
    for (int i = 0; i < 26; i++)
        if (count[i] % 2 == 1)
            res++;

    // As one character can be odd return
    // res - 1 but if string is already
    // palindrome return 0
    return (res == 0) ? 0 : res - 1;
}

// Driver program
int main()
{
    string str = "geeksforgeeks";
    cout << minInsertion(str);
```

```
    return 0;
}
```

Java

```
 // Java program to find minimum number
// of insertions to make a string
// palindrome
public class Palindrome {

    // Function will return number of
    // characters to be added
    static int minInsertion(String str)
    {
        // To store string length
        int n = str.length();

        // To store number of characters
        // occurring odd number of times
        int res = 0;

        // To store count of each
        // character
        int[] count = new int[26];

        // To store occurrence of each
        // character
        for (int i = 0; i < n; i++)
            count[str.charAt(i) - 'a']++;

        // To count characters with odd
        // occurrence
        for (int i = 0; i < 26; i++) {
            if (count[i] % 2 == 1)
                res++;
        }

        // As one character can be odd return
        // res - 1 but if string is already
        // palindrome return 0
        return (res == 0) ? 0 : res - 1;
    }

    // Driver program
    public static void main(String[] args)
    {
        String str = "geeksforgeeks";
        System.out.println(minInsertion(str));
```

```
    }
}
```

C#

```
 // C# program to find minimum number
// of insertions to make a string
// palindrome
using System;

public class GFG {

    // Function will return number of
    // characters to be added
    static int minInsertion(String str)
    {

        // To store string length
        int n = str.Length;

        // To store number of characters
        // occurring odd number of times
        int res = 0;

        // To store count of each
        // character
        int[] count = new int[26];

        // To store occurrence of each
        // character
        for (int i = 0; i < n; i++)
            count[str[i] - 'a']++;

        // To count characters with odd
        // occurrence
        for (int i = 0; i < 26; i++) {
            if (count[i] % 2 == 1)
                res++;
        }

        // As one character can be odd
        // return res - 1 but if string
        // is already palindrome
        // return 0
        return (res == 0) ? 0 : res - 1;
    }

    // Driver program
```

```
    public static void Main()
    {
        string str = "geeksforgeeks";

        Console.WriteLine(minInsertion(str));
    }
}

// This code is contributed by vt_m.
```

Output:

```
2
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/minimum-insertions-to-form-a-palindrome-with-permutations-allowed/

Chapter 212

Minimum length of jumps to avoid given array of obstacles

Minimum length of jumps to avoid given array of obstacles - GeeksforGeeks

We are given coordinates of obstacles on a straight line. We start jumping from point 0, we need to reach end avoiding all obstacles. Length of every jump has to be same (For example, if we jump from 0 to 4, then we must make next jump from 4 to 8). We need to find the minimum length of jump so that we can reach end and we avoid all obstacles.

Examples:

```
Input : obs[] = [5, 3, 6, 7, 9]
Output : 4
Obstacles are at points 3, 5, 6, 7 and 9
We jump from 0 to 4, then 4 to 8, then 4
to 12. This is how we reach end with jumps
of length 4. If we try lower jump lengths,
we cannot avoid all obstacles.

Input : obs[] = [5, 8, 9, 13, 14]
Output : 6
```

*

We insert locations of all obstacles in a hash table. We also find maximum value of obstacle. Then we try all possible jump sizes from 1 to maximum. If any jump size leads to a obstacle, we do not consider that jump.

```
 // Java program to find length of a jump
// to reach end avoiding all obstacles
import java.util.*;
```

```
public class obstacle {
    static int avoidObstacles(int[] obs)
    {
        // Insert all array elements in a hash table
        // and find the maximum value in the array
        HashSet<Integer> hs = new HashSet<Integer>();
        int max = obs[0];
        for (int i=0; i<obs.length; i++)
        {
            hs.add(obs[i]);
            max = Math.max(max, obs[i]);
        }

        // checking for every possible length which
        // yield us solution
        for (int i = 1; i <= max; i++) {
            int j;
            for (j = i; j <= max; j = j + i) {

                // if there is obstacle, break the loop.
                if (hs.contains(j))
                    break;
            }

            // If above loop did not break
            if (j > max)
                return i;
        }

        return max+1;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int a[] = new int[] { 5, 3, 6, 7, 9 };
        int b = avoidObstacles(a);
        System.out.println(b);
    }
}
```

Output:

```
4
```

Source

https://www.geeksforgeeks.org/minimum-length-of-jumps-to-avoid-given-array-of-obstacles/

Chapter 213

Minimum number of distinct elements after removing m items

Minimum number of distinct elements after removing m items - GeeksforGeeks

Given an array of items, an i-th index element denotes the item id's and given a number m, the task is to remove m elements such that there should be minimum distinct id's left.Print the number of distinct id's.

Examples:

```
Input : arr[] = { 2, 2, 1, 3, 3, 3}
            m = 3
Output : 1
Remove 1 and both 2's.So, only 3 will be
left that's why distinct id is 1.

Input : arr[] = { 2, 4, 1, 5, 3, 5, 1, 3}
            m = 2
Output : 3
Remove 2 and 4 completely. So, remaining ids
are 1, 3 and 5 i.e. 3
```

Asked in : Morgan Stanley

1- Count the occurrence of elements and store in the hash.
2- Sort the hash.
3- Start removing elements from hash.
4- Return the number of values left in the hash.

C++

```
 // C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to find distintc id's
int distinctIds(int arr[], int n, int mi)
{
    unordered_map<int, int> m;
    vector<pair<int, int> > v;
    int count = 0;

    // Store the occurrence of ids
    for (int i = 0; i < n; i++)
        m[arr[i]]++;

    // Store into the vector second as first and vice-versa
    for (auto it = m.begin(); it != m.end(); it++)
        v.push_back(make_pair(it->second, it->first));

    // Sort the vector
    sort(v.begin(), v.end());

    int size = v.size();

    // Start removing elements from the beginning
    for (int i = 0; i < size; i++) {

        // Remove if current value is less than
        // or equal to mi
        if (v[i].first <= mi) {
            mi -= v[i].first;
            count++;
        }

        // Return the remaining size
        else
            return size - count;
    }
    return size - count;
}

// Driver code
int main()
{
    int arr[] = { 2, 3, 1, 2, 3, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int m = 3;
```

```
    cout << distinctIds(arr, n, m);
    return 0;
}
```

Java

```
 //Java program for Minimum number of
//distinct elements after removing m items
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class DistinctIds
{
    // Function to find distintc id's
    static int distinctIds(int arr[], int n, int mi)
    {

        Map<Integer, Integer> m = new HashMap<Integer, Integer>();
        int count = 0;
        int size = 0;

        // Store the occurrence of ids
        for (int i = 0; i < n; i++)
        {

            // If the key is not add it to map
            if (m.containsKey(arr[i]) == false)
            {
                m.put(arr[i], 1);
                size++;
            }

            // If it is present then increase the value by 1
            else m.put(arr[i], m.get(arr[i]) + 1);
        }

        // Start removing elements from the beginning
        for (Entry<Integer, Integer> mp:m.entrySet())
        {
            // Remove if current value is less than
            // or equal to mi
            if (mp.getKey() <= mi)
            {
                mi -= mp.getKey();
                count++;
            }
```

```
            // Return the remaining size
            else return size - count;
        }

        return size - count;
    }

    //Driver method to test above function
    public static void main(String[] args)
    {
        // TODO Auto-generated method stub
        int arr[] = {2, 3, 1, 2, 3, 3};
        int m = 3;

        System.out.println(distinctIds(arr, arr.length, m));
    }
}
//This code is contributed by Sumit Ghosh
```

Output:

```
1
```

Time Complexity : O(n log n)

Source

https://www.geeksforgeeks.org/minimum-number-of-distinct-elements-after-removing-m-items/

Chapter 214

Minimum number of stops from given path

Minimum number of stops from given path - GeeksforGeeks

There are many points in two-dimensional space which need to be visited in a specific sequence. Path from one point to other is always chosen as shortest path and path segments are always aligned with grid lines. Now we are given the path which is chosen for visiting the points, we need to tell the minimum number of points that must be needed to generate given path.
Examples:

```
In above diagram, we can see that there
must be at least 3 points to get above
path, which are denoted by A, B and C
```

We can solve this problem by observing the pattern of movement when visiting the stops. If we want to take the shortest path from one point to another point then we will move in either one or max two directions i.e. it is always possible to reach the other point following maximum two directions and if more than two directions are used then that path won't be shortest, for example, path LLURD can be replaced with LLL only, so to find minimum number of stops in the path, we will loop over the characters of the path and maintain a map of directions taken till now. If at any index we found both 'L' as well as 'R' or we found both 'U' as well as 'D' then there must be a stop at current index, so we will increase the stop count by one and we will clear the map for next segment.

Total time complexity of the solution will be O(N)

```
 // C++ program to find minimum number of points
// in a given path
#include <bits/stdc++.h>
```

```
using namespace std;

// method returns minimum number of points in given path
int numberOfPointInPath(string path)
{
    int N = path.length();

    // Map to store last occurrence of direction
    map<char, int> dirMap;

    // variable to store count of points till now,
    // initializing from 1 to count first point
    int points = 1;

    // looping over all characters of path string
    for (int i = 0; i < N; i++) {

        // storing current direction in curDir
        // variable
        char curDir = path[i];

        // marking current direction as visited
        dirMap[curDir] = 1;

        // if at current index, we found both 'L'
        // and 'R' or 'U' and 'D' then current
        // index must be a point
        if ((dirMap['L'] && dirMap['R']) ||
            (dirMap['U'] && dirMap['D'])) {

            // clearing the map for next segment
            dirMap.clear();

            // increasing point count
            points++;

            // revisitng current direction for next segment
            dirMap[curDir] = 1;
        }
    }

    // +1 to count the last point also
    return (points + 1);
}

// Driver code to test above methods
int main()
{
```

```
    string path = "LLUUULLDD";
    cout << numberOfPointInPath(path) << endl;
    return 0;
}
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/minimum-number-stops-given-path/

Chapter 215

Minimum number of subsets with distinct elements

Minimum number of subsets with distinct elements - GeeksforGeeks

You are given an array of n-element. You have to make subsets from the array such that no subset contain duplicate elements. Find out minimum number of subset possible.

Examples :

```
Input : arr[] = {1, 2, 3, 4}
Output :1
Explanation : A single subset can contains all
values and all values are distinct

Input : arr[] = {1, 2, 3, 3}
Output : 2
Explanation : We need to create two subsets
{1, 2, 3} and {3} [or {1, 3} and {2, 3}] such
that both subsets have distinct elements.
```

We basically need to find the most frequent element in the array. The result is equal to the frequency of the most frequent element.

A **simple solution** is to run two nested loops to count frequency of every element and return the frequency of the most frequent element. Time complexity of this solution is $O(n^2)$.

A **better solution** is to first sort the array and then start count number of repetitions of elements in an iterative manner as all repetition of any number lie beside the number itself. By this method you can find the maximum frequency or repetition by simply traversing the sorted array. This approach will cost O(nlogn) time complexity

C++

```
 // A sorting based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
#include <bits/stdc++.h>
using namespace std;

// Function to count subsets such that all
// subsets have distinct elements.
int subset(int ar[], int n)
{
    // Take input and initialize res = 0
    int res = 0;

    // Sort the array
    sort(ar, ar + n);

    // Traverse the input array and
    // find maximum frequency
    for (int i = 0; i < n; i++) {
        int count = 1;

        // For each number find its repetition / frequency
        for (; i < n - 1; i++) {
            if (ar[i] == ar[i + 1])
                count++;
            else
                break;
        }

        // Update res
        res = max(res, count);
    }

    return res;
}

// Driver code
int main()
{
    int arr[] = { 5, 6, 9, 3, 4, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << subset(arr, n);
    return 0;
}
```

Java

```
 // A sorting based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
import java.util.*;
import java.lang.*;

public class GfG{

    // Function to count subsets such that all
    // subsets have distinct elements.
    public static int subset(int ar[], int n)
    {
        // Take input and initialize res = 0
        int res = 0;

        // Sort the array
        Arrays.sort(ar);

        // Traverse the input array and
        // find maximum frequency
        for (int i = 0; i < n; i++) {
            int count = 1;

            // For each number find its repetition / frequency
            for (; i < n - 1; i++) {
                if (ar[i] == ar[i + 1])
                    count++;
                else
                    break;
            }

            // Update res
            res = Math.max(res, count);
        }

        return res;
    }

    // Driver function
    public static void main(String argc[])
    {
        int arr[] = { 5, 6, 9, 3, 4, 3, 4 };
        int n = 7;
        System.out.println(subset(arr, n));
    }
```

```
}

/* This code is contributed by Sagar Shukla */
```

Python3

```
 # A sorting based solution to find the
# minimum number of subsets of a set
# such that every subset contains distinct
# elements.

# function to count subsets such that all
# subsets have distinct elements.
def subset(ar, n):

    # take input and initialize res = 0
    res = 0

    # sort the array
    ar.sort()

    # traverse the input array and
    # find maximum frequency
    for i in range(0, n) :
        count = 1

        # for each number find its repetition / frequency
        for i in range(n - 1):
            if ar[i] == ar[i + 1]:
                count+=1
            else:
                break

        # update res
        res = max(res, count)

    return res


# Driver code
ar = [ 5, 6, 9, 3, 4, 3, 4 ]
n = len(ar)
print(subset(ar, n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // A sorting based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
using System;

public class GfG {

    // Function to count subsets such that all
    // subsets have distinct elements.
    public static int subset(int []ar, int n)
    {
        // Take input and initialize res = 0
        int res = 0;

        // Sort the array
        Array.Sort(ar);

        // Traverse the input array and
        // find maximum frequency
        for (int i = 0; i < n; i++) {
            int count = 1;

            // For each number find its
            // repetition / frequency
            for ( ; i < n - 1; i++) {
                if (ar[i] == ar[i + 1])
                    count++;
                else
                    break;
            }

            // Update res
            res = Math.Max(res, count);
        }

        return res;
    }

    // Driver function
    public static void Main()
    {
        int []arr = { 5, 6, 9, 3, 4, 3, 4 };
        int n = 7;

        Console.WriteLine(subset(arr, n));
```

```
    }

}

/* This code is contributed by Vt_m */
```

Output :

```
2
```

An **efficient solution** is to use hashing. We count frequencies of all elements in a hash table. Finally we return the key with maximum value in hash table.

```
 // A hashing based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
#include <bits/stdc++.h>
using namespace std;

// Function to count subsets such that all
// subsets have distinct elements.
int subset(int arr[], int n)
{
    // Traverse the input array and
    // store frequencies of elements
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Find the maximum value in map.
    int res = 0;
    for (auto x : mp)
       res = max(res, x.second);

    return res;
}

// Driver code
int main()
{
    int arr[] = { 5, 6, 9, 3, 4, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << subset(arr, n);
    return 0;
}
```

Output :

2

Source

https://www.geeksforgeeks.org/minimum-number-subsets-distinct-elements/

Chapter 216

Minimum operation to make all elements equal in array

Minimum operation to make all elements equal in array - GeeksforGeeks

Given an array with n positive integers. We need to find the minimum number of operation to make all elements equal. We can perform addition, multiplication, subtraction or division with any element on an array element.

Examples:

```
Input : arr[] = {1, 2, 3, 4}
Output : 3
Since all elements are different,
we need to perform at-least three
operations to make them same. For
example, we can make them all 1
by doing three subtractions. Or make
them all 3 by doing three additions.

Input : arr[] = {1, 1, 1, 1}
Output : 0
```

For making all elements equal you can select a target value and then you can make all elements equal to that. Now, for converting a single element to target value you can perform a single operation only once. In this manner you can achieve your task in maximum of n operations but you have to minimize this number of operation and for this your selection of target is very important because if you select a target whose frequency in array is x then you have to perform only n-x more operations as you have already x elements equal to your target value. So, finally our task is reduced to finding element with maximum frequency. This can be achieved by different means such as iterative method in O(n^2), sorting in

O(nlogn) and hashing in O(n) time complexity.

C++

```
 // CPP program to find minimum number of
// operations required to make all elements
// of array equal
#include <bits/stdc++.h>
using namespace std;

// function for min operation
int minOperation (int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i=0; i<n; i++)
        hash[arr[i]]++;

    // find the max frequency
    int max_count = 0;
    for (auto i : hash)
        if (max_count < i.second)
            max_count = i.second;

    // return result
    return (n - max_count);
}

// driver program
int main()
{
    int arr[] = {1, 5, 2, 1, 3, 2, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << minOperation(arr, n);
    return 0;
}
```

Java

```
 // JAVA Code For Minimum operation to make
// all elements equal in array
import java.util.*;

class GFG {

    // function for min operation
    public static int minOperation (int arr[], int n)
```

```
    {
        // Insert all elements in hash.
       HashMap<Integer, Integer> hash = new HashMap<Integer,
                                            Integer>();

        for (int i=0; i<n; i++)
            if(hash.containsKey(arr[i]))
                hash.put(arr[i], hash.get(arr[i])+1);
            else hash.put(arr[i], 1);

        // find the max frequency
        int max_count = 0;
        Set<Integer> s = hash.keySet();

        for (int i : s)
            if (max_count < hash.get(i))
                max_count = hash.get(i);

        // return result
        return (n - max_count);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.length;
        System.out.print(minOperation(arr, n));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Output:

```
4
```

Source

https://www.geeksforgeeks.org/minimum-operation-make-elements-equal-array/

Chapter 217

Most frequent element in an array

Most frequent element in an array - GeeksforGeeks

Given an array, find the most frequent element in it. If there are multiple elements that appear maximum number of times, print any one of them.

Examples:

```
Input : arr[] = {1, 3, 2, 1, 4, 1}
Output : 1
1 appears three times in array which
is maximum frequency.

Input : arr[] = {10, 20, 10, 20, 30, 20, 20}
Output : 20
```

A **simple solution** is to run two loops. The outer loop picks all elements one by one. The inner loop finds frequency of the picked element and compares with the maximum so far. Time complexity of this solution is $O(n^2)$

A **better solution** is to do sorting. We first sort the array, then linearly traverse the array.

C++

```
 // CPP program to find the most frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int mostFrequent(int arr[], int n)
{
```

```
    // Sort the array
    sort(arr, arr + n);

    // find the max frequency using linear traversal
    int max_count = 1, res = arr[0], curr_count = 1;
    for (int i = 1; i < n; i++) {
        if (arr[i] == arr[i - 1])
            curr_count++;
        else {
            if (curr_count > max_count) {
                max_count = curr_count;
                res = arr[i - 1];
            }
            curr_count = 1;
        }
    }

    // If last element is most frequent
    if (curr_count > max_count)
    {
        max_count = curr_count;
        res = arr[n - 1];
    }

    return res;
}

// driver program
int main()
{
    int arr[] = { 1, 5, 2, 1, 3, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << mostFrequent(arr, n);
    return 0;
}
```

Java

```
 //Java program to find the most frequent element
//in an array
import java.util.*;

class GFG {

    static int mostFrequent(int arr[], int n)
    {

        // Sort the array
```

```
        Arrays.sort(arr);

        // find the max frequency using linear
        // traversal
        int max_count = 1, res = arr[0];
        int curr_count = 1;

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1])
                curr_count++;
            else
            {
                if (curr_count > max_count)
                {
                    max_count = curr_count;
                    res = arr[i - 1];
                }
                curr_count = 1;
            }
        }

        // If last element is most frequent
        if (curr_count > max_count)
        {
            max_count = curr_count;
            res = arr[n - 1];
        }

        return res;
    }

    // Driver program
    public static void main (String[] args) {

        int arr[] = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.length;

        System.out.println(mostFrequent(arr,n));

    }
}

// This code is contributed by Akash Singh.
```

Python3

```
 # Python3 program to find the most
```

```
# frequent element in an array.

def mostFrequent(arr, n):

    # Sort the array
    arr.sort()

    # find the max frequency using
    # linear traversal
    max_count = 1; res = arr[0]; curr_count = 1

    for i in range(1, n):
        if (arr[i] == arr[i - 1]):
            curr_count += 1

        else :
            if (curr_count > max_count):
                max_count = curr_count
                res = arr[i - 1]

            curr_count = 1

    # If last element is most frequent
    if (curr_count > max_count):

        max_count = curr_count
        res = arr[n - 1]

    return res

# Driver Code
arr = [1, 5, 2, 1, 3, 2, 1]
n = len(arr)
print(mostFrequent(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find the most
// frequent element in an array
using System;

class GFG {

    static int mostFrequent(int []arr, int n)
    {
```

```
        // Sort the array
        Array.Sort(arr);

        // find the max frequency using
        // linear traversal
        int max_count = 1, res = arr[0];
        int curr_count = 1;

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1])
                curr_count++;
            else
            {
                if (curr_count > max_count)
                {
                    max_count = curr_count;
                    res = arr[i - 1];
                }
                curr_count = 1;
            }
        }

        // If last element is most frequent
        if (curr_count > max_count)
        {
            max_count = curr_count;
            res = arr[n - 1];
        }

        return res;
    }

    // Driver code
    public static void Main ()
    {

        int []arr = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.Length;

        Console.WriteLine(mostFrequent(arr,n));

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find the
// most frequent element
// in an array.

function mostFrequent( $arr, $n)
{

    // Sort the array
    sort($arr);
    sort($arr , $n);

    // find the max frequency
    // using linear traversal
    $max_count = 1;
    $res = $arr[0];
    $curr_count = 1;
    for ($i = 1; $i < $n; $i++)
    {
        if ($arr[$i] == $arr[$i - 1])
            $curr_count++;
        else
        {
            if ($curr_count > $max_count)
            {
                $max_count = $curr_count;
                $res = $arr[$i - 1];
            }
            $curr_count = 1;
        }
    }

    // If last element
    // is most frequent
    if ($curr_count > $max_count)
    {
        $max_count = $curr_count;
        $res = $arr[$n - 1];
    }

    return $res;
}

// Driver Code
{
    $arr = array(1, 5, 2, 1, 3, 2, 1);
    $n = sizeof($arr) / sizeof($arr[0]);
    echo mostFrequent($arr, $n);
```

```
    return 0;
}

// This code is contributed by nitin mittal
?>
```

Output :

```
1
```

Time Complexity : O(n Log n)
Auxiliary Space : O(1)

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key value pairs. Finally we traverse the hash table and print the key with maximum value.

C++

```
 // CPP program to find the most frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int mostFrequent(int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++)
        hash[arr[i]]++;

    // find the max frequency
    int max_count = 0, res = -1;
    for (auto i : hash) {
        if (max_count < i.second) {
            res = i.first;
            max_count = i.second;
        }
    }

    return res;
}

// driver program
int main()
{
```

```
    int arr[] = { 1, 5, 2, 1, 3, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << mostFrequent(arr, n);
    return 0;
}
```

Java

```
 //Java program to find the most frequent element
//in an array
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

class GFG {

    static int mostFrequent(int arr[], int n)
    {

        // Insert all elements in hash
        Map<Integer, Integer> hp =
               new HashMap<Integer, Integer>();

        for(int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(hp.containsKey(key))
            {
                int freq = hp.get(key);
                freq++;
                hp.put(key, freq);
            }
            else
            {
                hp.put(key, 1);
            }
        }

        // find max frequency.
        int max_count = 0, res = -1;

        for(Entry<Integer, Integer> val : hp.entrySet())
        {
            if (max_count < val.getValue())
            {
                res = val.getKey();
                max_count = val.getValue();
            }
```

```
        }

        return res;
    }

    // Driver code
    public static void main (String[] args) {

        int arr[] = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.length;

        System.out.println(mostFrequent(arr, n));
    }
}

// This code is contributed by Akash Singh.
```

C#

```
 // C# program to find the most
// frequent element in an array
using System;
using System.Collections.Generic;

class GFG
{
    static int mostFrequent(int []arr,
                            int n)
    {
        // Insert all elements in hash
        Dictionary<int, int> hp =
                    new Dictionary<int, int>();

        for (int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(hp.ContainsKey(key))
            {
                int freq = hp[key];
                freq++;
                hp[key] = freq;
            }
            else
                hp.Add(key, 1);
        }

        // find max frequency.
        int min_count = 0, res = -1;
```

```
        foreach (KeyValuePair<int,
                    int> pair in hp)
        {
            if (min_count < pair.Value)
            {
                res = pair.Key;
                min_count = pair.Value;
            }
        }
        return res;
    }

    // Driver code
    static void Main ()
    {
        int []arr = new int[]{1, 5, 2,
                              1, 3, 2, 1};
        int n = arr.Length;

        Console.Write(mostFrequent(arr, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
1
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : vt_m, nitin mittal, dipesh_jain, manishshaw1

Source

https://www.geeksforgeeks.org/frequent-element-array/

Chapter 218

Most frequent word in an array of strings

Most frequent word in an array of strings - GeeksforGeeks

Given an array of words find the most occurring word in it

Examples:

```
Input : arr[] = {"geeks", "for", "geeks", "a",
                "portal", "to", "learn", "can",
                "be", "computer", "science",
                 "zoom", "yup", "fire", "in",
                 "be", "data"}
Output : Geeks
"geeks" is the most frequent word as it
occurs 3 times
```

A **simple solution** is to run two loops and count occurrences of every word. Time complexity of this solution is O(n * n * MAX_WORD_LEN).

An **efficient solution** is to use Trie data structure. The idea is simple first we will insert in trie. In trie, we keep counts of words ending at a node. We do preorder traversal and compare count present at each node and find the maximum occurring word

```
 // CPP code to find most frequent word in
// an array of strings
#include <bits/stdc++.h>
using namespace std;

/*structing the trie*/
struct Trie {
```

```
    string key;
    int cnt;
    unordered_map<char, Trie*> map;
};

/* Function to return a new Trie node */
Trie* getNewTrieNode()
{
    Trie* node = new Trie;
    node->cnt = 0;
    return node;
}

/* function to insert a string */
void insert(Trie*& root, string &str)
{
    // start from root node
    Trie* temp = root;

    for (int i=0; i<str.length(); i++) {

        char x = str[i];

        /*a new node if path doesn't exists*/
        if (temp->map.find(x) == temp->map.end())
            temp->map[x] = getNewTrieNode();

        // go to next node
        temp = temp->map[x];
    }

    // store key and its count in leaf nodes
    temp->key = str;
    temp->cnt += 1;
}

/* function for preorder traversal */
bool preorder(Trie* temp, int& maxcnt, string& key)
{
    if (temp == NULL)
        return false;

    for (auto it : temp->map) {

        /*leaf node will have non-zero count*/
        if (maxcnt < it.second->cnt) {
            key = it.second->key;
            maxcnt = it.second->cnt;
```

```
        }

        // recurse for current node children
        preorder(it.second, maxcnt, key);
    }
}

void mostFrequentWord(string arr[], int n)
{
    // Insert all words in a Trie
    Trie* root = getNewTrieNode();
    for (int i = 0; i < n; i++)
        insert(root, arr[i]);

    // Do preorder traversal to find the
    // most frequent word
    string key;
    int cnt = 0;
    preorder(root, cnt, key);

    cout << "The word that occurs most is : "
        << key << endl;
    cout << "No of times: " << cnt << endl;
}

// Driver code
int main()
{
    // given set of keys
    string arr[] = {"geeks", "for", "geeks", "a",
            "portal", "to", "learn", "can", "be",
            "computer", "science", "zoom", "yup",
            "fire", "in", "be", "data", "geeks"};
    int n = sizeof(arr) / sizeof(arr[0]);

    mostFrequentWord(arr, n);

    return 0;
}
```

Output:

```
The word that occurs most is : geeks
No of times: 3
```

Time Complexity : O(n * MAX_WORD_LEN)

Another efficient solution is to use hashing. Please refer Find winner of an election where votes are represented as candidate names for details.

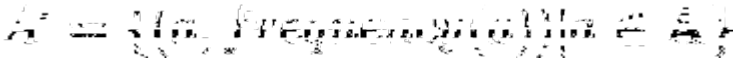
Source

https://www.geeksforgeeks.org/frequent-word-array-strings/

Chapter 219

Multiset Equivalence Problem

Multiset Equivalence Problem - GeeksforGeeks

Unlike a set, a multiset may contain multiple occurrences of same number. The *multiset* equivalence problem states to check if two given multisets are equal or not. For example let **A** = {1, 2, 3} and **B** = {1, 1, 2, 3}. Here **A** is set but **B** is not (1 occurs twice in B), whereas **A** and **B** are both multisets. More formally, "Are the *sets* of pairs defined as  equal for the two given multisets?"

Given two multisets A and B, write a program to check if the two multisets are equal.

Note: Elements in the multisets can be of order 10^9

Examples:

```
Input : A = {1, 1, 3, 4},
        B = {1, 1, 3, 4}
Output : Yes

Input : A = {1, 3},
        B = {1, 1}
Output : No
```

Since the elements are as large as 10^9 we cannot use direct index table.

One solution is to sort both multisets and compare them one by one.

```
 // C++ program to check if two given multisets
// are equivalent
#include <bits/stdc++.h>
using namespace std;
```

```
bool areSame(vector<int>& a, vector<int>& b)
{
    // sort the elements of both multisets
    sort(a.begin(), a.end());
    sort(b.begin(), b.end());

    // Return true if both multisets are same.
    return (a == b);
}

int main()
{
    vector<int> a({ 7, 7, 5 }), b({ 7, 5, 5 });
    if (areSame(a, b))
        cout << "Yes\n";
    else
        cout << "No\n";
    return 0;
}
```

Output:

```
No
```

A better solution is to use hashing. We create two empty hash tables (implemented using unordered_map in C++). We first insert all items of first multimap in first table and all items of second multiset in second table. Now we check if both hash tables contain same items and frequencies or not.

```
 // C++ program to check if two given multisets
// are equivalent
#include <bits/stdc++.h>
using namespace std;

bool areSame(vector<int>& a, vector<int>& b)
{
    if (a.size() != b.size())
        return false;

    // Create two unordered maps m1 and m2
    // and insert values of both vectors.
    unordered_map<int, int> m1, m2;
    for (int i = 0; i < a.size(); i++) {
        m1[a[i]]++;
        m2[b[i]]++;
    }
```

```
    // Now we check if both unordered_maps
    // are same of not.
    for (auto x : m1) {
        if (m2.find(x.first) == m2.end() ||
            m2[x.first] != x.second)
            return false;
    }

    return true;
}

// Driver code
int main()
{
    vector<int> a({ 7, 7, 5 }), b({ 7, 7, 5 });
    if (areSame(a, b))
        cout << "Yes\n";
    else
        cout << "No\n";
    return 0;
}
```

Output:

```
Yes
```

Time complexity : O(n) under the assumption that unordered_map find() and insert() operations work in O(1) time.

Source

https://www.geeksforgeeks.org/multiset-equivalence-problem/

Chapter 220

Next Greater Frequency Element

Next Greater Frequency Element - GeeksforGeeks

Given an array, for each element find the value of nearest element to the right which is having frequency greater than as that of current element. If there does not exist an answer for a position, then make the value '-1'.

Examples:

```
Input : a[] = [1, 1, 2, 3, 4, 2, 1]
Output : [-1, -1, 1, 2, 2, 1, -1]
Explanation:
Given array a[] = [1, 1, 2, 3, 4, 2, 1]
Frequency of each element is: 3, 3, 2, 1, 1, 2, 3
Lets calls Next Greater Frequency element as NGF
1. For element a[0] = 1 which has a frequency = 3,
   As it has frequency of 3 and no other next element
   has frequency more than 3 so  '-1'
2. For element a[1] = 1 it will be -1 same logic
   like a[0]
3. For element a[2] = 2 which has frequency = 2,
   NGF element is 1 at position = 6  with frequency
   of 3 > 2
4. For element a[3] = 3 which has frequency = 1,
   NGF element is 2 at position = 5 with frequency
   of 2 > 1
5. For element a[4] = 4 which has frequency = 1,
   NGF element is 2 at position = 5 with frequency
   of 2 > 1
6. For element a[5] = 2 which has frequency = 2,
```

```
   NGF element is 1 at position = 6 with frequency
   of 3 > 2
7. For element a[6] = 1 there is no element to its
   right, hence -1

Input : a[] = [1, 1, 1, 2, 2, 2, 2, 11, 3, 3]
Output : [2, 2, 2, -1, -1, -1, -1, 3, -1, -1]
```

Naive approach:
A simple hashing technique is to use values as index is be used to store frequency of each element. Create a list suppose to store frequency of each number in the array. (Single traversal is required). Now use two loops.
The outer loop picks all the elements one by one.
The inner loop looks for the first element whose frequency is greater than the frequency of current element.
If a greater frequency element is found then that element is printed, otherwise -1 is printed.
Time complexity : O(n*n)

Efficient approach:
We can use hashing and stack data structure to efficiently solve for many cases. A simple hashing technique is to use values as index and frequency of each element as value. We use stack data structure to store position of elements in the array.

1) Create a list to to use values as index to store frequency of each element.
2) Push the position of first element to stack.
3) Pick rest of the position of elements one by one and follow following steps in loop.
…….**a)** Mark the position of current element as 'i' .
……. **b)** If the frequency of the element which is pointed by the top of stack is **greater** than frequency of the current element, push the current position i to the stack
……. **c)** If the frequency of the element which is pointed by the top of stack is **less** than frequency of the current element and the stack is not empty then follow these steps:
…….***i)*** continue popping the stack
…….***ii)*** if the condition in step c fails then push the current position i to the stack
4) After the loop in step 3 is over, pop all the elements from stack and print -1 as next greater frequency element for them does not exist.

Time complexity is O(n).

Below is the Python 3 implementation of the above problem.

```
 '''NFG function to find the next greater frequency
   element for each element in the array'''
def NFG(a, n):

    if (n <= 0):
```

```
    print("List empty")
    return []

# stack data structure to store the position
# of array element
stack = [0]*n

# freq is a dictionary which maintains the
# frequency of each element
freq = {}
for i in a:
    freq[a[i]] = 0
for i in a:
    freq[a[i]] += 1

# res to store the value of next greater
# frequency element for each element
res = [0]*n

# initialize top of stack to -1
top = -1

# push the first position of array in the stack
top += 1
stack[top] = 0

# now iterate for the rest of elements
for i in range(1, n):

    ''' If the frequency of the element which is
        pointed by the top of stack is greater
        than frequency of the current element
        then push the current position i in stack'''
    if (freq[a[stack[top]]] > freq[a[i]]):
        top += 1
        stack[top] = i

    else:
        ''' If the frequency of the element which
        is pointed by the top of stack is less
        than frequency of the current element, then
        pop the stack and continuing popping until
        the above condition is true while the stack
        is not empty'''

        while (top>-1 and freq[a[stack[top]]] < freq[a[i]]):
            res[stack[top]] = a[i]
            top -= 1
```

```
            # now push the current element
            top+=1
            stack[top] = i

    '''After iterating over the loop, the remaining
    position of elements in stack do not have the
    next greater element, so print -1 for them'''
    while (top > -1):
        res[stack[top]] = -1
        top -= 1

    # return the res list containing next
    # greater frequency element
    return res

# Driver program to test the function
print(NFG([1,1,2,3,4,2,1],7))
```

Output:

```
[-1, -1, 1, 2, 2, 1, -1]
```

Source

https://www.geeksforgeeks.org/next-greater-frequency-element/

Chapter 221

Non-Repeating Element

Non-Repeating Element - GeeksforGeeks

Find the first non-repeating element in a given array of integers.

Examples:

```
Input : -1 2 -1 3 2
Output : 3
Explanation : The first number that does not
repeat is : 3

Input : 9 4 9 6 7 4
Output : 6
```

A **Simple Solution** is to use two loops. The outer loop picks elements one by one and inner loop checks if the element is present more than once or not.

C++

```
 // Simple CPP program to find first non-
// repeating element.
#include <bits/stdc++.h>
using namespace std;

int firstNonRepeating(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
        int j;
        for (j=0; j<n; j++)
            if (i != j && arr[i] == arr[j])
                break;
        if (j == n)
```

```
            return arr[i];
    }
    return -1;
}

// Driver code
int main()
{
    int arr[] = { 9, 4, 9, 6, 7, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << firstNonRepeating(arr, n);
    return 0;
}
```

Java

```
 // Java program to find first non-repeating
// element.
class GFG {

    static int firstNonRepeating(int arr[], int n)
    {
        for (int i = 0; i < n; i++) {
            int j;
            for (j = 0; j < n; j++)
                if (i != j && arr[i] == arr[j])
                    break;
            if (j == n)
                return arr[i];
        }

        return -1;
    }

    //Driver code
    public static void main (String[] args)
    {

        int arr[] = { 9, 4, 9, 6, 7, 4 };
        int n = arr.length;

        System.out.print(firstNonRepeating(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find first
# non-repeating element.

def firstNonRepeating(arr, n):

    for i in range(n):
        j = 0
        while(j < n):
            if (i != j and arr[i] == arr[j]):
                break
            j += 1
        if (j == n):
            return arr[i]

    return -1

# Driver code
arr = [ 9, 4, 9, 6, 7, 4 ]
n = len(arr)
print(firstNonRepeating(arr, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find first non-
// repeating element.
using System;

class GFG
{
    static int firstNonRepeating(int []arr, int n)
    {
        for (int i = 0; i < n; i++) {
            int j;
            for (j = 0; j < n; j++)
                if (i != j && arr[i] == arr[j])
                    break;
            if (j == n)
                return arr[i];
        }
        return -1;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = { 9, 4, 9, 6, 7, 4 };
```

```
        int n = arr.Length;
        Console.Write(firstNonRepeating(arr, n));
    }
}
// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// Simple PHP program to find first non-
// repeating element.

function firstNonRepeating($arr, $n)
{
    for ($i = 0; $i < $n; $i++)
    {
        $j;
        for ($j = 0; $j< $n; $j++)
            if ($i != $j && $arr[$i] == $arr[$j])
                break;
        if ($j == $n)
            return $arr[$i];
    }
    return -1;
}

    // Driver code
    $arr = array(9, 4, 9, 6, 7, 4);
    $n = sizeof($arr) ;
    echo firstNonRepeating($arr, $n);

// This code is contributed by ajit
?>
```

Output:

```
6
```

An **Efficient Solution** is to use hashing.
1) Traverse array and insert elements and their counts in hash table.
2) Traverse array again and print first element with count equals to 1.

```
 // Efficient CPP program to find first non-
// repeating element.
#include <bits/stdc++.h>
```

```
using namespace std;

int firstNonRepeating(int arr[], int n)
{
    // Insert all array elements in hash
    // table
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
       mp[arr[i]]++;

    // Traverse array again and return
    // first element with count 1.
    for (int i = 0; i < n; i++)
       if (mp[arr[i]] == 1)
            return arr[i];
    return -1;
}

// Driver code
int main()
{
    int arr[] = { 9, 4, 9, 6, 7, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << firstNonRepeating(arr, n);
    return 0;
}
```

Output:

```
6
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Further Optimization: If array has many duplicates, we can also store index in hash table, using a hash table where value is a pair. Now we only need to traverse keys in hash table (not complete array) to find first non repeating.

Printing all non-repeating elements:

```
 // Efficient CPP program to print all non-
// repeating elements.
#include <bits/stdc++.h>
using namespace std;

void firstNonRepeating(int arr[], int n)
{
```

```
    // Insert all array elements in hash
    // table
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
       mp[arr[i]]++;

    // Traverse through map only and
    for (auto x : mp)
       if (x.second == 1)
            cout << x.first << " ";
}

// Driver code
int main()
{
    int arr[] = { 9, 4, 9, 6, 7, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    firstNonRepeating(arr, n);
    return 0;
}
```

Output:

```
7 6
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/non-repeating-element/

Chapter 222

Non-overlapping sum of two sets

Non-overlapping sum of two sets - GeeksforGeeks

Given two arrays A[] and B[] of size n. It is given that both array individually contain distinct elements. We need to find sum of all elements that are not common.

Examples:

```
Input : A[] = {1, 5, 3, 8}
        B[] = {5, 4, 6, 7}
Output : 29
1 + 3 + 4 + 6 + 7 + 8 = 29

Input : A[] = {1, 5, 3, 8}
        B[] = {5, 1, 8, 3}
Output : 0
All elements are common.
```

Brute Force Method :
One simple approach is that for each element in A[] check whether it is present in B[], if it is present in then add it to the result. Similarly traverse B[] and for every element that is not present in B, add it to result.
Time Complexity: O(n^2).

Hashing concept :
Create an empty hash and insert elements of both arrays into it. Now traverse hash table and add all those elements whose count is 1. (As per the question, both arrays individually have distinct elements)

Below is the implementation of above approach:

```
 // CPP program to find Non-overlapping sum
#include <bits/stdc++.h>
```

```
using namespace std;


// function for calculating
// Non-overlapping sum of two array
int findSum(int A[], int B[], int n)
{
    // Insert elements of both arrays
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++) {
        hash[A[i]]++;
        hash[B[i]]++;
    }

    // calculate non-overlapped sum
    int sum = 0;
    for (auto x: hash)
        if (x.second == 1)
            sum += x.first;

    return sum;
}

// driver code
int main()
{
    int A[] = { 5, 4, 9, 2, 3 };
    int B[] = { 2, 8, 7, 6, 3 };

    // size of array
    int n = sizeof(A) / sizeof(A[0]);

    // function call
    cout << findSum(A, B, n);
    return 0;
}
```

Output:

```
39
```

Source

https://www.geeksforgeeks.org/overlapping-sum-two-array/

Chapter 223

Number of Counterclockwise shifts to make a string palindrome

Number of Counterclockwise shifts to make a string palindrome - GeeksforGeeks

Given a string of lowercase English alphabets, find the number of counterclockwise shifts of characters required to make the string palindrome. It is given that shifting the string will always result in the palindrome.

Examples:

> **Input:** str = “baabbccb”
> **Output:** 2
> Shifting the string counter clockwise 2 times,
> will make the string palindrome.
> 1st shift : aabbccbb
> 2nd shift :abbccbba
>
> **Input:** bbaabbcc
> **Output**: 3
> Shifting the string counter clockwise
> 3 times will make the string palindrome.
> 1st shift : baabbccb
> 2nd shift : aabbccbb
> 3rd shift : abbccbba

Naive Approach: A naive approach is to one by one shift character of the given string counter-clockwise cyclically and check if the string is palindrome or not.

Better Approach: A better approach is to append the string with itself and iterate from the first character to the last character of the given string. The substring from i to i+n (where i is in range [0,n-1]) in the appended string will be the string obtained after every

counterclockwise shift. Check for the substring if it is palindrome or not. The number of shift operations will be i.

Below is the implementation of above approach:

```
 // C++ program to find counter clockwise
// shifts to make string palindrome.
#include <bits/stdc++.h>
using namespace std;

// Function to check if given string is
// palindrome or not.
bool isPalindrome(string str, int l, int r)
{
    while (l < r) {
        if (str[l] != str[r])
            return false;

        l++;
        r--;
    }

    return true;
}

// Function to find counter clockwise shifts
// to make string palindrome.
int CyclicShifts(string str)
{

    int n = str.length();

    // Pointer to starting of current
    // shifted string.
    int left = 0;

    // Pointer to ending of current
    // shifted string.
    int right = n - 1;

    // Concatenate string with itself
    str = str + str;

    // To store counterclockwise shifts
    int cnt = 0;

    // Move left and right pointers one
    // step at a time.
    while (right < 2 * n - 1) {
```

```
        // Check if current shifted string
        // is palindrome or not
        if (isPalindrome(str, left, right))
            break;

        // If string is not palindrome
        // then increase count of number
        // of shifts by 1.
        cnt++;

        left++;
        right++;
    }

    return cnt;
}

// Driver code.
int main()
{
    string str = "bccbbaab";

    cout << CyclicShifts(str);
    return 0;
}
```

Output:

```
2
```

Time Complexity: O(N^2)
Auxiliary Space: O(N)

Efficient Approach: An efficient approach is to use Cumulative Hash. The string is shifted cyclically according to the method explained above and the hash value of this string is compared to the hash value of the reversed string. If both values are same then current shifted string is palindrome otherwise string is again shifted. The count of shifts will be i at any step. To calculate value of both strings below hash function is used:

H(s) = $(31^i * (S_i - \text{'a'}))$ % mod, 0 i (length of string – 1)
where, H(x) = Hash function
s = given string
mod = $10^9 + 7$

Iterate for all the substrings and check for if it is a palindrome or not using the hash function stated above and the cumulative hash technique.

Below is the implementation of above approach:

```
 // CPP program to find counter clockwise
// shifts to make string palindrome.
#include <bits/stdc++.h>

#define mod 1000000007
using namespace std;

// Function to find counter clockwise shifts
// to make string palindrome.
int CyclicShifts(string str)
{

    int n = str.length(), i;

    // To store power of 31.
    // po[i] = 31^i;
    long long int po[2 * n + 2];

    // To store hash value of string.
    long long int preval[2 * n + 2];

    // To store hash value of reversed
    // string.
    long long int suffval[2 * n + 2];

    // To find hash value of string str[i..j]
    long long int val1;

    // To store hash value of reversed string
    // str[j..i]
    long long int val2;

    // To store number of counter clockwise
    // shifts.
    int cnt = 0;

    // Concatenate string with itself to shift
    // it cyclically.
    str = str + str;

    // Calculate powers of 31 upto 2*n which
    // will be used in hash function.
    po[0] = 1;
    for (i = 1; i <= 2 * n; i++) {
        po[i] = (po[i - 1] * 31) % mod;
    }
```

```
    // Hash value of string str[0..i] is stored in
    // preval[i].
    for (i = 1; i <= 2 * n; i++) {
        preval[i] = ((preval[i - 1] * 31) % mod +
                     (str[i - 1] - 'a')) % mod;
    }

    // Hash value of string str[i..n-1] is stored
    // in suffval[i].
    for (i = 2 * n; i > 0; i--) {
        suffval[i] = ((suffval[i + 1] * 31) % mod +
                       (str[i - 1] - 'a')) % mod;
    }

    // Characters in string str[0..i] is present
    // at position [(n-1-i)..(n-1)] in reversed
    // string. If hash value of both are same
    // then string is palindrome else not.
    for (i = 1; i <= n; i++) {

        // Hash value of shifted string starting at
        // index i and ending at index i+n-1.
        val1 = (preval[i + n - 1] -
               ((po[n] * preval[i - 1]) % mod)) % mod;
        if (val1 < 0)
            val1 += mod;

        // Hash value of corresponding string when
        // reversed starting at index i+n-1 and
        // ending at index i.
        val2 = (suffval[i] - ((po[n] * suffval[i + n])
                                       % mod)) % mod;
        if (val2 < 0)
            val2 += mod;

        // If both hash value are same then current
        // string str[i..(i+n-1)] is palindrome.
        // Else increase the shift count.
        if (val1 != val2)
            cnt++;
        else
            break;
    }

    return cnt;
}
```

```
// Driver code.
int main()
{
    string str = "bccbbaab";

    cout << CyclicShifts(str);
    return 0;
}
```

Output:

```
2
```

Time Complexity: O(N)
Auxiliary Space: O(N)

Source

https://www.geeksforgeeks.org/number-of-counterclockwise-shifts-to-make-a-string-palindrome/

Chapter 224

Number of GP (Geometric Progression) subsequences of size 3

Number of GP (Geometric Progression) subsequences of size 3 - GeeksforGeeks

Given n elements and a ratio r, find the number of G.P. subsequences with length 3. A subsequence is considered GP with length 3 with ration r.

Examples:

```
Input : arr[] = {1, 1, 2, 2, 4}
            r = 2
Output : 4
Explanation: Any of the two 1s can be chosen
as the first element, the second element can
be any of the two 2s, and the third element
of the subsequence must be equal to 4.

Input : arr[] = {1, 1, 2, 2, 4}
            r = 3
Output : 0
```

A **naive approach** is to use three nested for loops and check for every subsequence with length 3 and keep a count of the subsequences. The complexity is $O(n^3)$.

An **efficient approach** is to solve the problem for fixed middle element of progression. This means that if we fix element a[i] as middle, then it must be multiple of r, and a[i]/r and a[i]*r must be present. We count number of occurrences of a[i]/r and a[i]*r and then multiply the counts. To do this, we can use concept of hashing where we store the count of all possible

elements in two hash maps, one indicating the number of elements in the left and the other indicating the number of elements to the right.

Below is the c++ implementation of the above approach

```
 // CPP program to count GP subsequences of size 3.
#include <bits/stdc++.h>
using namespace std;

// Returns count of G.P. subseqeunces
// with length 3 and common ratio r
long long subsequences(int a[], int n, int r)
{
    // hashing to maintain left and right array
    // elements to the main count
    unordered_map<int, int> left, right;

    // stores the answer
    long long ans = 0;

    // traverse through the elements
    for (int i = 0; i < n; i++)
        right[a[i]]++; // keep the count in the hash

    // traverse through all elements
    // and find out the number of elements as k1*k2
    for (int i = 0; i < n; i++) {

        // keep the count of left and right elements
        // left is a[i]/r and right a[i]*r
        long long c1 = 0, c2 = 0;

        // if the current element is divisible by k,
        // count elements in left hash.
        if (a[i] % r == 0)
            c1 = left[a[i] / r];

        // decrease the count in right hash
        right[a[i]]--;

        // number of right elements
        c2 = right[a[i] * r];

        // calculate the answer
        ans += c1 * c2;

        left[a[i]]++; // left count of a[i]
    }
```

```
    // returns answer
    return ans;
}

// driver program
int main()
{
    int a[] = { 1, 2, 6, 2, 3, 6, 9, 18, 3, 9 };
    int n = sizeof(a) / sizeof(a[0]);
    int r = 3;
    cout << subsequences(a, n, r);
    return 0;
}
```

Output:

```
6
```

Time complexity: O(n)

Source

https://www.geeksforgeeks.org/number-gp-geometric-progression-subsequences-size-3/

Chapter 225

Number of NGEs to the right

Number of NGEs to the right - GeeksforGeeks

Given an array of n integers and q queries, print the number of next greater elements to the right of the given index element.

Examples:

```
Input : a[] = {3, 4, 2, 7, 5, 8, 10, 6}
              q = 2
              index = 0,
              index = 5
Output: 4
        1
Explanation: the next greater elements
to the right of 3(index 0) are 4, 7, 8,
10. The next greater elements to the right
of 8(index 5) are 10.
```

A **naive approach** is to iterate for every query from index to end, and find out the number of next greater elements to the right. This won't be efficient enough as we run two nested loops .

Time complexity: O(n) to answer a query.
Auxiliary space: O(1)

Better approach is to store the next greater index of every element and run a loop for every query that iterates from index and keeping the increasing counter as j = next[i]. This will avoid checking all elements and will directly jump to the next greater element of every element. But this won't be efficient enough in cases like 1 2 3 4 5 6, where the next greater elements are sequentially increasing, ending it up in taking O(n) for every query.

Time complexity : O(n) to answer a query.
Auxiliary space : O(n) for next greater element.

Efficient approach is to store the next greater elements index using next greater element in a next[] array. Then create a dp[] array that starts from n-2, as n-1th index will have no elements to its right and dp[n-1] = 0. While traversing from back we use dynamic programming to count the number of elements to the right where we use memoization as dp[next[i]] which gives us a count of the numbers to the right of the next greater element of the current element, hence we add 1 to it. If next[i]=-1 then we do not have any element to the right hence dp[i]=0. dp[index] stores the count of the number of next greater elements to the right.

Below is the c++ implementation of the above approach

```
 #include <bits/stdc++.h>
using namespace std;

// array to store the next greater element index
void fillNext(int next[], int a[], int n)
{
    // use of stl stack in c++
    stack<int> s;

    // push the 0th index to the stack
    s.push(0);

    // traverse in the loop from 1-nth index
    for (int i = 1; i < n; i++) {

        // iterate till loop is empty
        while (!s.empty()) {

            // get the topmost index in the stack
            int cur = s.top();

            // if the current element is greater
            // then the top index-th element, then
            // this will be the next greatest index
            // of the top index-th element
            if (a[cur] < a[i]) {

                // initialize the cur index position's
                // next greatest as index
                next[cur] = i;

                // pop the cur index as its greater
                // element has been found
                s.pop();
            }

            // if not greater then break
            else
```

```
                break;
        }

        // push the i index so that its next greatest
        // can be found
        s.push(i);
    }

    // iterate for all other index left inside stack
    while (!s.empty()) {

        int cur = s.top();

        // mark it as -1 as no element in greater
        // then it in right
        next[cur] = -1;

        s.pop();
    }
}

// function to count the number of next greater numbers to the right
void count(int a[], int dp[], int n)
{
    // initializes the next array as 0
    int next[n];
    memset(next, 0, sizeof(next));

    // calls the function to pre-calculate
    // the next greatest element indexes
    fillNext(next, a, n);

    for (int i = n - 2; i >= 0; i--) {

        // if the i-th element has no next
        // greater element to right
        if (next[i] == -1)
            dp[i] = 0;

        // Count of next greater numbers to right.
        else
            dp[i] = 1 + dp[next[i]];
    }
}

// answers all queries in O(1)
int answerQuery(int dp[], int index)
{
```

```
    // returns the number of next greater
    // elements to the right.
    return dp[index];
}

// driver program to test the above function
int main()
{
    int a[] = { 3, 4, 2, 7, 5, 8, 10, 6 };
    int n = sizeof(a) / sizeof(a[0]);

    int dp[n];

    // calls the function to count the number
    // of greater elements to the right for
    // every element.
    count(a, dp, n);

    // query 1 answered
    cout << answerQuery(dp, 3) << endl;

    // query 2 answered
    cout << answerQuery(dp, 6) << endl;

    // query 3 answered
    cout << answerQuery(dp, 1) << endl;

    return 0;
}
```

Output:

```
2
0
3
```

Time complexity: O(1) to answer a query.
Auxiliary Space: O(n)

Source

https://www.geeksforgeeks.org/number-nges-right/

Chapter 226

Number of ordered points pair satisfying line equation

Number of ordered points pair satisfying line equation - GeeksforGeeks

Given an array of n integers, slope of a line i. e., m and the intercept of the line i.e c, Count the number of ordered pairs(i, j) of points where i j, such that point (A_i, A_j) satisfies the line formed with given slope and intercept.
Note : The equation of the line is y = mx + c, where m is the slope of the line and c is the intercept.

Examples :

> Input : m = 1, c = 1, arr[] = [1, 2, 3, 4, 2]
> Output : 5 ordered points
> **Explanation :** The equation of the line with given slope and intercept is : y = x + 1. The Number of pairs (i, j), for which (arr_i, arr_j) satisfies the above equation of the line are : (1, 2), (1, 5), (2, 3), (3, 4), (5, 3).
>
> Input : m = 2, c = 1, arr[] = [1, 2, 3, 4, 2, 5]
> Output : 3 ordered points

Method 1 (Brute Force):
Generate all possible pairs (i, j) and check if a particular ordered pair (i, j) is such that, (arr_i, arr_j) satisfies the given equation of the line y = mx + c, and i j. If the point is valid(a point is valid if the above condition is satisfied), increment the counter which stores the total number of valid points.

Source

https://www.geeksforgeeks.org/number-ordered-points-pair-satisfying-line-equation/

C++

```
 // CPP code to count the number of ordered
// pairs satisfying Line Equation
#include <bits/stdc++.h>

using namespace std;

/* Checks if (i, j) is valid, a point (i, j)
   is valid if point (arr[i], arr[j])
   satisfies the equation y = mx + c And
   i is not equal to j*/
bool isValid(int arr[], int i, int j,
             int m, int c)
{

    // check if i equals to j
    if (i == j)
        return false;


    // Equation LHS = y, and RHS = mx + c
    int lhs = arr[j];
    int rhs = m * arr[i] + c;

    return (lhs == rhs);
}

/* Returns the number of ordered pairs
   (i, j) for which point (arr[i], arr[j])
   satisfies the equation of the line
   y = mx + c */
int findOrderedPoints(int arr[], int n,
                      int m, int c)
{

    int counter = 0;

    // for every possible (i, j) check
    // if (a[i], a[j]) satisfies the
    // equation y = mx + c
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            // (firstIndex, secondIndex)
            // is same as (i, j)
            int firstIndex = i, secondIndex = j;

            // check if (firstIndex,
```

```
            // secondIndex) is a valid point
            if (isValid(arr, firstIndex, secondIndex, m, c))
                counter++;
        }
    }
    return counter;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // equation of line is y = mx + c
    int m = 1, c = 1;
    cout << findOrderedPoints(arr, n, m, c);
    return 0;
}
```

Java

```
 // Java code to find number of ordered
// points satisfying line equation
import java.io.*;

public class GFG {

    // Checks if (i, j) is valid,
    // a point (i, j) is valid if
    // point (arr[i], arr[j])
    // satisfies the equation
    // y = mx + c And
    // i is not equal to j
    static boolean isValid(int []arr, int i,
                           int j, int m, int c)
    {

        // check if i equals to j
        if (i == j)
            return false;


        // Equation LHS = y,
        // and RHS = mx + c
        int lhs = arr[j];
        int rhs = m * arr[i] + c;
```

```
    return (lhs == rhs);
}

/* Returns the number of ordered pairs
(i, j) for which point (arr[i], arr[j])
satisfies the equation of the line
y = mx + c */
static int findOrderedPoints(int []arr,
                  int n, int m, int c)
{

    int counter = 0;

    // for every possible (i, j) check
    // if (a[i], a[j]) satisfies the
    // equation y = mx + c
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {

            // (firstIndex, secondIndex)
            // is same as (i, j)
            int firstIndex = i,
               secondIndex = j;

            // check if (firstIndex,
            // secondIndex) is a
            // valid point
            if (isValid(arr, firstIndex,
                     secondIndex, m, c))
                counter++;
        }
    }
    return counter;
}

// Driver Code
public static void main(String args[])
{
    int []arr = { 1, 2, 3, 4, 2 };
    int n = arr.length;

    // equation of line is y = mx + c
    int m = 1, c = 1;
    System.out.print(
       findOrderedPoints(arr, n, m, c));
}
```

```
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
 # Python code to count the number of ordered
# pairs satisfying Line Equation

# Checks if (i, j) is valid, a point (i, j)
# is valid if point (arr[i], arr[j])
# satisfies the equation y = mx + c And
# i is not equal to j
def isValid(arr, i, j, m, c) :

    # check if i equals to j
    if (i == j) :
        return False


    # Equation LHS = y, and RHS = mx + c
    lhs = arr[j];
    rhs = m * arr[i] + c

    return (lhs == rhs)

# Returns the number of ordered pairs
# (i, j) for which point (arr[i], arr[j])
# satisfies the equation of the line
# y = mx + c */
def findOrderedPoints(arr, n, m, c) :

    counter = 0

    # for every possible (i, j) check
    # if (a[i], a[j]) satisfies the
    # equation y = mx + c
    for i in range(0, n) :
        for j in range(0, n) :
            # (firstIndex, secondIndex)
            # is same as (i, j)
            firstIndex = i
            secondIndex = j

            # check if (firstIndex,
            # secondIndex) is a valid point
            if (isValid(arr, firstIndex,
```

```
                    secondIndex, m, c)) :
                counter = counter + 1

    return counter

# Driver Code
arr = [ 1, 2, 3, 4, 2 ]
n = len(arr)

# equation of line is y = mx + c
m = 1
c = 1
print (findOrderedPoints(arr, n, m, c))

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
 // C# code to find number of ordered
// points satisfying line equation
using System;
class GFG {

    // Checks if (i, j) is valid,
    // a point (i, j) is valid if
    // point (arr[i], arr[j])
    // satisfies the equation
    // y = mx + c And
    // i is not equal to j
    static bool isValid(int []arr, int i,
                        int j, int m, int c)
    {

        // check if i equals to j
        if (i == j)
            return false;


        // Equation LHS = y,
        // and RHS = mx + c
        int lhs = arr[j];
        int rhs = m * arr[i] + c;

        return (lhs == rhs);
    }

    /* Returns the number of ordered pairs
```

```
      (i, j) for which point (arr[i], arr[j])
      satisfies the equation of the line
       y = mx + c */
    static int findOrderedPoints(int []arr, int n,
                                 int m, int c)
    {

        int counter = 0;

        // for every possible (i, j) check
        // if (a[i], a[j]) satisfies the
        // equation y = mx + c
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {

                // (firstIndex, secondIndex)
                // is same as (i, j)
                int firstIndex = i, secondIndex = j;

                // check if (firstIndex,
                // secondIndex) is a valid point
                if (isValid(arr, firstIndex, secondIndex, m, c))
                    counter++;
            }
        }
        return counter;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = { 1, 2, 3, 4, 2 };
        int n = arr.Length;

        // equation of line is y = mx + c
        int m = 1, c = 1;
        Console.Write(findOrderedPoints(arr, n, m, c));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
```

```
// PHP code to count the
// number of ordered pairs
// satisfying Line Equation

/* Checks if (i, j) is valid,
a point (i, j) is valid if
point (arr[i], arr[j]) satisfies
the equation y = mx + c And i
is not equal to j*/
function isValid($arr, $i,
                 $j, $m, $c)
{
    // check if i equals to j
    if ($i == $j)
        return false;

    // Equation LHS = y, and
    // RHS = mx + c
    $lhs = $arr[$j];
    $rhs = $m * $arr[$i] + $c;

    return ($lhs == $rhs);
}

/* Returns the number of
ordered pairs (i, j) for
which point (arr[i], arr[j])
satisfies the equation of
the line y = mx + c */
function findOrderedPoints($arr, $n,
                           $m, $c)
{

    $counter = 0;

    // for every possible (i, j)
    // check if (a[i], a[j])
    // satisfies the equation
    // y = mx + c
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $n; $j++)
        {
            // (firstIndex, secondIndex)
            // is same as (i, j)
            $firstIndex = $i; $secondIndex = $j;

            // check if (firstIndex,
```

```
            // secondIndex) is a valid point
            if (isValid($arr, $firstIndex,
                        $secondIndex, $m, $c))
                $counter++;
        }
    }
    return $counter;
}

// Driver Code
$arr = array( 1, 2, 3, 4, 2 );
$n = count($arr);

// equation of line
// is y = mx + c
$m = 1; $c = 1;
echo (findOrderedPoints($arr, $n, $m, $c));

// This code is contributed by
// Manish Shaw (manishshaw1)
?>
```

Output :

```
5
```

Time Complexity : $O(n^2)$

Method 2 (Efficient) :
Given a x coordinate of a point, for each x there is a unique value of y and the value of y is nothing but m * x + c. So, for each possible x coordinate of the array arr, calculate how many times the unique value of y which satisfies the equation of the line occurs in that array. Store count of all the integers of array, arr in a map. Now, for each value, arr_i, add to the answer, the number of occurrences of m * arr_i + c. For a given i, m * a[i] + c occurs x times in the array, then, add x to our counter for total valid points, but need to check that if a[i] = m * a[i] + c then, it is obvious that since this occurs x times in the array then one occurrence is at the i^{th} index and rest (x – 1) occurrences are the valid y coordinates so add (x – 1) to our points counter.

C++

```
 // CPP code to find number of ordered
// points satisfying line equation
#include <bits/stdc++.h>
using namespace std;

/* Returns the number of ordered pairs
```

```
   (i, j) for which point (arr[i], arr[j])
   satisfies the equation of the line
   y = mx + c */
int findOrderedPoints(int arr[], int n,
                      int m, int c)
{
    int counter = 0;

    // map stores the frequency
    // of arr[i] for all i
    unordered_map<int, int> frequency;

    for (int i = 0; i < n; i++)
        frequency[arr[i]]++;

    for (int i = 0; i < n; i++)
    {
        int xCoordinate = arr[i];
        int yCoordinate = (m * arr[i] + c);

        // if for a[i](xCoordinate),
        // a yCoordinate exists in the map
        // add the frequency of yCoordinate
        // to the counter

        if (frequency.find(yCoordinate) !=
            frequency.end())
            counter += frequency[yCoordinate];

        // check if xCoordinate = yCoordinate,
        // if this is true then since we only
        // want (i, j) such that i != j, decrement
        // the counter by one to avoid points
        // of type (arr[i], arr[i])
        if (xCoordinate == yCoordinate)
            counter--;
    }
    return counter;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 1, c = 1;
    cout << findOrderedPoints(arr, n, m, c);
    return 0;
```

```
}
```

Output:

```
5
```

Time Complexity : O(n)

Improved By : manishshaw1

Chapter 227

Number of subarrays having sum exactly equal to k

Number of subarrays having sum exactly equal to k - GeeksforGeeks

Given an unsorted array of integers, find number of subarrays having sum exactly equal to a given number k.

Examples:

```
Input : arr[] = {10, 2, -2, -20, 10},
        k = -10
Output : 3
Subarrays: arr[0...3], arr[1...4], arr[3..4]
have sum exactly equal to -10.

Input : arr[] = {9, 4, 20, 3, 10, 5},
            k = 33
Output : 2
Subarrays : arr[0...2], arr[2...4] have sum
exactly equal to 33.
```

A **simple** solution is to traverse all the subarrays and calculate their sum. If sum is equal to the required sum then increment the count of subarrays. Print final count of subarrays.

An **efficient** solution is while traversing the array, store sum so far in currsum. Also maintain count of different values of currsum in a map. If value of currsum is equal to desired sum at any instance increment count of subarrays by one. The value of currsum exceeds desired sum by currsum – sum. If this value is removed from currsum then desired sum can be obtained. From the map find number of subarrays previously found having sum equal to currsum-sum. Excluding all those subarrays from current subarray, gives new subarrays having desired sum. So increase count by the number of such subarrays. Note that when

currsum is equal to desired sum then also check number of subarrays previously having sum equal to 0. Excluding those subarrays from current subarray gives new subarrays having desired sum. Increase count by the number of subarrays having sum 0 in that case.

```
 // C++ program to find number of subarrays
// with sum exactly equal to k.
#include <bits/stdc++.h>
using namespace std;

// Function to find number of subarrays
// with sum exactly equal to k.
int findSubarraySum(int arr[], int n, int sum)
{
    // STL map to store number of subarrays
    // starting from index zero having
    // particular value of sum.
    unordered_map<int, int> prevSum;

    int res = 0;

    // Sum of elements so far.
    int currsum = 0;

    for (int i = 0; i < n; i++) {

        // Add current element to sum so far.
        currsum += arr[i];

        // If currsum is equal to desired sum,
        // then a new subarray is found. So
        // increase count of subarrays.
        if (currsum == sum)
            res++;

        // currsum exceeds given sum by currsum
        //  - sum. Find number of subarrays having
        // this sum and exclude those subarrays
        // from currsum by increasing count by
        // same amount.
        if (prevSum.find(currsum - sum) !=
                                  prevSum.end())
            res += (prevSum[currsum - sum]);


        // Add currsum value to count of
        // different values of sum.
        prevSum[currsum]++;
    }
```

```
    return res;
}

int main()
{
    int arr[] = { 10, 2, -2, -20, 10 };
    int sum = -10;
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findSubarraySum(arr, n, sum);
    return 0;
}
```

Output:

```
3
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Source

https://www.geeksforgeeks.org/number-subarrays-sum-exactly-equal-k/

Chapter 228

Numbers having Unique (or Distinct) digits

Numbers having Unique (or Distinct) digits - GeeksforGeeks

Given a range, print all numbers having unique digits.

Examples :

```
Input : 10 20
Output : 10 12 13 14 15 16 17 18 19 20  (Except 11)

Input : 1 10
Output : 1 2 3 4 5 6 7 8 9 10
```

```
As the problem is pretty simple, the only thing to be done is :-
1- Find the digits one by one and keep marking visited digits.
2- If all digits occurs one time only then print that number.
3- Else not.
```

C++

```
 // C++ implementation to find unique digit
// numbers in a range
#include<bits/stdc++.h>
using namespace std;

// Function to print unique digit numbers
// in range from l to r.
void printUnique(int l, int r)
```

```
{
    // Start traversing the numbers
    for (int i=l ; i<=r ; i++)
    {
        int num = i;
        bool visited[10] = {false};

        // Find digits and maintain its hash
        while (num)
        {
            // if a digit occcurs more than 1 time
            // then break
            if (visited[num % 10])
                break;

            visited[num%10] = true;

            num = num/10;
        }

        // num will be 0 only when above loop
        // doesn't get break that means the
        // number is unique so print it.
        if (num == 0)
            cout << i << " ";
    }
}

// Driver code
int main()
{
    int l = 1, r = 20;
    printUnique(l, r);
    return 0;
}
```

Java

```
 // Java implementation to find unique digit
// numbers in a range
class Test
{
    // Method to print unique digit numbers
    // in range from l to r.
    static void printUnique(int l, int r)
    {
        // Start traversing the numbers
        for (int i=l ; i<=r ; i++)
```

```
        {
            int num = i;
            boolean visited[] = new boolean[10];

            // Find digits and maintain its hash
            while (num != 0)
            {
                // if a digit occcurs more than 1 time
                // then break
                if (visited[num % 10])
                    break;

                visited[num%10] = true;

                num = num/10;
            }

            // num will be 0 only when above loop
            // doesn't get break that means the
            // number is unique so print it.
            if (num == 0)
                System.out.print(i + " ");
        }
    }

    // Driver method
    public static void main(String args[])
    {
        int l = 1, r = 20;
        printUnique(l, r);
    }
}
```

Python3

```
 # Python3 implementation
# to find unique digit
# numbers in a range

# Function to print
# unique digit numbers
# in range from l to r.
def printUnique(l,r):

    # Start traversing
    # the numbers
    for i in range (l, r + 1):
        num = i;
```

```
        visited = [0,0,0,0,0,0,0,0,0,0];

        # Find digits and
        # maintain its hash
        while (num):

            # if a digit occcurs
            # more than 1 time
            # then break
            if visited[num % 10] == 1:
                break;
            visited[num % 10] = 1;
            num = (int)(num / 10);

        # num will be 0 only when
        # above loop doesn't get
        # break that means the
        # number is unique so
        # print it.
        if num == 0:
            print(i, end = " ");

# Driver code
l = 1;
r = 20;
printUnique(l, r);

# This code is
# contributed by mits
```

C#

```
 // C# implementation to find unique digit
// numbers in a range
using System;

public class GFG {

    // Method to print unique digit numbers
    // in range from l to r.
    static void printUnique(int l, int r)
    {

        // Start traversing the numbers
        for (int i = l ; i <= r ; i++)
        {
            int num = i;
            bool []visited = new bool[10];
```

```
            // Find digits and maintain
            // its hash
            while (num != 0)
            {

                // if a digit occcurs more
                // than 1 time then break
                if (visited[num % 10])
                    break;

                visited[num % 10] = true;

                num = num / 10;
            }

            // num will be 0 only when
            // above loop doesn't get
            // break that means the number
            // is unique so print it.
            if (num == 0)
                Console.Write(i + " ");
        }
    }

    // Driver method
    public static void Main()
    {
        int l = 1, r = 20;
        printUnique(l, r);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP implementation to find unique
// digit numbers in a range

// Function to print unique digit
// numbers in range from l to r.
function printUnique($l, $r)
{
    // Start traversing the numbers
    for ($i = $l ; $i <= $r; $i++)
    {
```

```
        $num = $i;
        $visited = (false);

        // Find digits and
        // maintain its hash
        while ($num)
        {
            // if a digit occcurs more
            // than 1 time then break
            if ($visited[$num % 10])

            $visited[$num % 10] = true;

            $num = (int)$num / 10;
        }

        // num will be 0 only when above
        // loop doesn't get break that
        // means the number is unique
        // so print it.
        if ($num == 0)
            echo $i , " ";
    }
}

// Driver code
$l = 1; $r = 20;
printUnique($l, $r);

// This code is contributed by aj_36
?>
```

Output :

```
1 2 3 4 5 6 7 8 9 10 12 13 14 15 16 17 18 19 20
```

Improved By : Sam007, jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/numbers-unique-distinct-digits/

Chapter 229

Numbers with prime frequencies greater than or equal to k

Numbers with prime frequencies greater than or equal to k - GeeksforGeeks

Given an array, find elements which appears prime number of times in the array with minimum k frequency (frequency >= k).

Examples :

```
Input : int[] arr = { 11, 11, 11, 23, 11, 37, 51,
                      37, 37, 51, 51, 51, 51 };
        k = 2
Output : 37, 51
Explanation :
11's count is 4, 23 count 1, 37 count 3, 51 count 5.
37 and 51 are two number that appear prime number of
time and frequencies greater than or equal to k.

Input : int[] arr = { 11, 22, 33 } min Occurrence = 1
Output : -1
None of the count is prime number of times
```

Approach :
1. Create a Map which holds the number as Key and value as its occurrences in the input array.
2. Iterate the Map keys and look for the values corresponding to their keys, return the key which
has minimum value fulfilling condition key's value is a prime number and >= min occurrence provided
as input.

C++

```
 // C++ code to find number
// occurring prime number
// of times with frequency >= k
#include <bits/stdc++.h>
using namespace std;

// Check if the number of
// occurrences are primes
// or not
bool isPrime(int n)
{
    // Corner case
    if (n <= 1) return false;

    // Check from 2 to n-1
    for (int i = 2; i < n; i++)
        if (n % i == 0)
            return false;

    return true;
}

// Function to find number
// with prime occurrences
void primeOccurences(int arr[], int k)
{
    unordered_map<int, int> map;

    // Insert values and
    // their frequencies
    for (int i = 0; i < 12; i++)
        map[arr[i]]++;

    // Traverse map and find
    // elements with prime
    // frequencies and frequency
    // at least k
    for (auto x : map)
    {
        if (isPrime(x.second) &&
                    x.second >= k)
            cout << x.first << endl;
    }
}

// Driver code
int main()
{
```

```
    int arr[] = {11, 11, 11, 23,
                 11, 37, 37, 51,
                 51, 51, 51, 51};
    int k = 2;

    primeOccurences(arr, k);
    return 0;
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Java

```
 // Java code to find number occurring prime
// number of times with frequency >= k
import java.util.*;

public class PrimeNumber {

    // Function to find number with prime occurrences
    static void primeOccurences(int[] arr, int k)
    {
        Map<Integer, Integer> map = new HashMap<>();

        // Insert values and their frequencies
        for (int i = 0; i < arr.length; i++) {
            int val = arr[i];

            int freq;
            if (map.containsKey(val)) {
                freq = map.get(val);
                freq++;
            }
            else
                freq = 1;
            map.put(val, freq);
        }

        // Traverse map and find elements with
        // prime frequencies and frequency at
        // least k
        for (Map.Entry<Integer, Integer> entry :
                              map.entrySet()) {
            int value = entry.getValue();
            if (isPrime(value) && value >= k)
                System.out.println(entry.getKey());
        }
```

```
    }

    // Check if the number of occurrences
    // are primes or not
    private static boolean isPrime(int n)
    {

        if ((n > 2 && n % 2 == 0) || n == 1)
            return false;

        for (int i = 3; i <= (int)Math.sqrt(n);
             i += 2) {
            if (n % i == 0)
                return false;
        }
        return true;
    }

    // Driver code
    public static void main(String[] args)
    {
        int[] arr = { 11, 11, 11, 23, 11, 37,
                      37, 51, 51, 51, 51, 51 };
        int k = 2;

        primeOccurences(arr, k);
    }
}
```

Python3

```
 # Python3 code to find number
# occurring prime number of
# times with frequency >= k

# Function to find number
# with prime occurrences
def primeOccurences(arr, k):
    map = {}

    # Insert values and their frequencies
    for val in arr:
        freq = 0

        if val in map :
            freq = map[val]
            freq += 1
```

```
        else :
            freq = 1
        map[val] = freq

    # Traverse map and find elements
    # with prime frequencies and
    # frequency at least k
    for entry in map :
        value = map[entry]

        if isPrime(value) and value >= k:
            print(entry)

# Check if the number of occurrences
# are primes or not
def isPrime(n):

    if (n > 2 and not n % 2) or n == 1:
        return False

    for i in range(3, int(n**0.5 + 1), 2):
        if not n % i:
            return False

    return True


# Driver code

arr = [ 11, 11, 11, 23, 11, 37,
        37, 51, 51, 51, 51, 51 ]
k = 2

primeOccurences(arr, k)


# This code is contributed by Ansu Kumari.
```

C#

```
 // C# code to find number
// occurring prime number
// of times with frequency >= k
using System;
using System.Collections.Generic;

class GFG
{
```

```
// Function to find number
// with prime occurrences
static void primeOccurences(int[] arr,
                            int k)
{
    Dictionary<int, int> map =
               new Dictionary<int, int>();

    // Insert values and
    // their frequencies
    for (int i = 0; i < arr.Length; i++)
    {
        int val = arr[i];

        int freq;
        if (map.ContainsKey(val))
        {
            freq = map[val];
            freq++;
            map.Remove(val);
        }
        else
            freq = 1;
        map.Add(val, freq);
    }

    // Traverse map and find elements
    // with prime frequencies and
    // frequency atleast k
    foreach (KeyValuePair<int, int>
                      pair in map)
    {
        int value = pair.Value;
        if (isPrime(value) &&
                    value >= k)
            Console.WriteLine(pair.Key);
    }
}

// Check if the number
// of occurrences
// are primes or not
static bool isPrime(int n)
{
    if ((n > 2 &&
         n % 2 == 0) || n == 1)
        return false;
```

```
        for (int i = 3;
                 i <= (int)Math.Sqrt(n);
                 i += 2)
            {
                if (n % i == 0)
                    return false;
            }
        return true;
    }

    // Driver code
    static void Main()
    {
        int[] arr = new int[]{11, 11, 11, 23, 11, 37,
                              37, 51, 51, 51, 51, 51};
        int k = 2;

        primeOccurences(arr, k);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
37
51
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/numbers-prime-frequencies-greater-equal-k/

Chapter 230

Numbers with sum of digits equal to the sum of digits of its all prime factor

Numbers with sum of digits equal to the sum of digits of its all prime factor - GeeksforGeeks

Given a range, the task is to find the count of the numbers in the given range such that the sum of its digit is equal to the sum of all its prime factors digits sum.

Examples:

```
Input: l = 2, r = 10
Output: 5
2, 3, 4, 5 and 7 are such numbers

Input: l = 15, r = 22
Output: 3
17, 19 and 22 are such numbers
As, 17 and 19 are already prime.
Prime Factors of 22 = 2 * 11 i.e
For 22, Sum of digits is 2+2 = 4
For 2 * 11, Sum of digits is 2 + 1 + 1 = 4
```

Approach: An efficient solution is to modify Sieve of Eratosthenes such that for each non-prime number it stores smallest prime factor(prefactor).

1. Preprocess to find the smallest prime factorfor all the numbers between 2 and MAXN. This can be done by breaking up the number into its prime factors in constant time because for each number if it is a prime, it has no prefactor.

2. Otherwise, we can break it up to into a prime factor and the other part of the number which may or may not be prime.
3. And repeat this process of extracting factors till it becomes a prime.
4. Then check if the digits of that number is equal to the digits of prime factors by adding th digits of smallest prime factor i.e.

   Digits_Sum of SPF[n] + Digits_Sum of (n / SPF[n])

5. Now make prefix sum array that counts how many valid numbers are there up to a number N. For each query, print:

   ans[R] − ans[L-1]

Below is the implementation of above approach:

C++

```
 // C++ program to Find the count of the numbers
// in the given range such that the sum of its
// digit is equal to the sum of all its prime
// factors digits sum.
#include <bits/stdc++.h>
using namespace std;

// maximum size of number
#define MAXN 100005

// array to store smallest prime factor of number
int spf[MAXN] = { 0 };

// array to store sum of digits of a number
int sum_digits[MAXN] = { 0 };

// boolean array to check given number is countable
// for required answer or not.
bool isValid[MAXN] = { 0 };

// prefix array to store answer
int ans[MAXN] = { 0 };

// Calculating SPF (Smallest Prime Factor) for every
// number till MAXN.
void Smallest_prime_factor()
{
```

```
    // marking smallest prime factor for every
    // number to be itself.
    for (int i = 1; i < MAXN; i++)
        spf[i] = i;

    // separately marking spf for every even
    // number as 2
    for (int i = 4; i < MAXN; i += 2)
        spf[i] = 2;

    for (int i = 3; i * i <= MAXN; i += 2)

        // checking if i is prime
        if (spf[i] == i)

            // marking SPF for all numbers divisible by i
            for (int j = i * i; j < MAXN; j += i)

                // marking spf[j] if it is not
                // previously marked
                if (spf[j] == j)
                    spf[j] = i;
}

// Function to find sum of digits in a number
int Digit_Sum(int copy)
{
    int d = 0;
    while (copy) {
        d += copy % 10;
        copy /= 10;
    }

    return d;
}

// find sum of digits of all numbers up to MAXN
void Sum_Of_All_Digits()
{
    for (int n = 2; n < MAXN; n++) {
        // add sum of digits of least
        // prime factor and n/spf[n]
        sum_digits[n] = sum_digits[n / spf[n]]
                            + Digit_Sum(spf[n]);

        // if it is valid make isValid true
        if (Digit_Sum(n) == sum_digits[n])
            isValid[n] = true;
```

```
    }

    // prefix sum to compute answer
    for (int n = 2; n < MAXN; n++) {
        if (isValid[n])
            ans[n] = 1;
        ans[n] += ans[n - 1];
    }
}

// Driver code
int main()
{
    Smallest_prime_factor();
    Sum_Of_All_Digits();

    // decleartion
    int l, r;

    // print answer for required range
    l = 2, r = 3;
    cout << "Valid numbers in the range " << l << " "
         << r << " are " << ans[r] - ans[l - 1] << endl;

    // print answer for required range
    l = 2, r = 10;
    cout << "Valid numbers in the range " << l << " "
         << r << " are " << ans[r] - ans[l - 1] << endl;

  return 0;
}
```

Java

```
 // Java program to Find the count
// of the numbers in the given
// range such that the sum of its
// digit is equal to the sum of
// all its prime factors digits sum.
import java.io.*;

class GFG
{

// maximum size of number
static int MAXN = 100005;

// array to store smallest
```

```
// prime factor of number
static int spf[] = new int[MAXN];

// array to store sum
// of digits of a number
static int sum_digits[] = new int[MAXN];

// boolean array to check
// given number is countable
// for required answer or not.
static boolean isValid[] = new boolean[MAXN];

// prefix array to store answer
static int ans[] = new int[MAXN];

// Calculating SPF (Smallest
// Prime Factor) for every
// number till MAXN.
static void Smallest_prime_factor()
{
    // marking smallest prime factor
    // for every number to be itself.
    for (int i = 1; i < MAXN; i++)
        spf[i] = i;

    // separately marking spf
    // for every even number as 2
    for (int i = 4; i < MAXN; i += 2)
        spf[i] = 2;

    for (int i = 3;
             i * i <= MAXN; i += 2)

        // checking if i is prime
        if (spf[i] == i)

            // marking SPF for all
            // numbers divisible by i
            for (int j = i * i;
                     j < MAXN; j += i)

                // marking spf[j] if it
                // is not previously marked
                if (spf[j] == j)
                    spf[j] = i;
}

// Function to find sum
```

```
// of digits in a number
static int Digit_Sum(int copy)
{
    int d = 0;
    while (copy > 0)
    {
        d += copy % 10;
        copy /= 10;
    }

    return d;
}

// find sum of digits of
// all numbers up to MAXN
static void Sum_Of_All_Digits()
{
    for (int n = 2; n < MAXN; n++)
    {
        // add sum of digits of least
        // prime factor and n/spf[n]
        sum_digits[n] = sum_digits[n / spf[n]]
                          + Digit_Sum(spf[n]);

        // if it is valid make isValid true
        if (Digit_Sum(n) == sum_digits[n])
            isValid[n] = true;
    }

    // prefix sum to compute answer
    for (int n = 2; n < MAXN; n++)
    {
        if (isValid[n])
            ans[n] = 1;
        ans[n] += ans[n - 1];
    }
}

// Driver code
public static void main (String[] args)
{
    Smallest_prime_factor();
    Sum_Of_All_Digits();

    // declaration
    int l, r;

    // print answer for required range
```

```
    l = 2; r = 3;
    System.out.println("Valid numbers in the range " +
                               l + " " + r + " are " +
                              (ans[r] - ans[l - 1] ));

    // print answer for required range
    l = 2; r = 10;
    System.out.println("Valid numbers in the range " +
                               l + " " + r + " are " +
                               (ans[r] - ans[l - 1]));
}
}

// This code is contributed
// by Inder
```

C#

```
 // C# program to Find the count
// of the numbers in the given
// range such that the sum of its
// digit is equal to the sum of
// all its prime factors digits sum.
using System;

class GFG
{

// maximum size of number
static int MAXN = 100005;

// array to store smallest
// prime factor of number
static int []spf = new int[MAXN];

// array to store sum
// of digits of a number
static int []sum_digits = new int[MAXN];

// boolean array to check
// given number is countable
// for required answer or not.
static bool []isValid = new bool[MAXN];

// prefix array to store answer
static int []ans = new int[MAXN];

// Calculating SPF (Smallest
```

```
// Prime Factor) for every
// number till MAXN.
static void Smallest_prime_factor()
{
    // marking smallest prime factor
    // for every number to be itself.
    for (int i = 1; i < MAXN; i++)
        spf[i] = i;

    // separately marking spf
    // for every even number as 2
    for (int i = 4; i < MAXN; i += 2)
        spf[i] = 2;

    for (int i = 3;
             i * i <= MAXN; i += 2)

        // checking if i is prime
        if (spf[i] == i)

            // marking SPF for all
            // numbers divisible by i
            for (int j = i * i;
                     j < MAXN; j += i)

                // marking spf[j] if it
                // is not previously marked
                if (spf[j] == j)
                    spf[j] = i;
}

// Function to find sum
// of digits in a number
static int Digit_Sum(int copy)
{
    int d = 0;
    while (copy > 0)
    {
        d += copy % 10;
        copy /= 10;
    }

    return d;
}

// find sum of digits of
// all numbers up to MAXN
static void Sum_Of_All_Digits()
```

```
{
    for (int n = 2; n < MAXN; n++)
    {
        // add sum of digits of least
        // prime factor and n/spf[n]
        sum_digits[n] = sum_digits[n / spf[n]] +
                              Digit_Sum(spf[n]);

        // if it is valid make
        // isValid true
        if (Digit_Sum(n) == sum_digits[n])
            isValid[n] = true;
    }

    // prefix sum to compute answer
    for (int n = 2; n < MAXN; n++)
    {
        if (isValid[n])
            ans[n] = 1;
        ans[n] += ans[n - 1];
    }
}

// Driver code
public static void Main ()
{
    Smallest_prime_factor();
    Sum_Of_All_Digits();

    // declaration
    int l, r;

    // print answer for required range
    l = 2; r = 3;
    Console.WriteLine("Valid numbers in the range " +
                               l + " " + r + " are " +
                              (ans[r] - ans[l - 1] ));

    // print answer for required range
    l = 2; r = 10;
    Console.WriteLine("Valid numbers in the range " +
                               l + " " + r + " are " +
                               (ans[r] - ans[l - 1]));
}
}

// This code is contributed
// by Subhadeep
```

Output:

```
Valid numbers in the range 2 3 are 2
Valid numbers in the range 2 10 are 5
```

Improved By : inderDuMCA, tufan_gupta2000

Source

https://www.geeksforgeeks.org/numbers-with-sum-of-digits-equal-to-the-sum-of-digits-of-its-all-prime-factor/

Chapter 231

Nuts & Bolts Problem (Lock & Key problem) | Set 2 (Hashmap)

Nuts & Bolts Problem (Lock & Key problem) | Set 2 (Hashmap) - GeeksforGeeks

Given a set of n nuts of different sizes and n bolts of different sizes. There is a one-one mapping between nuts and bolts. Match nuts and bolts efficiently.
Constraint: Comparison of a nut to another nut or a bolt to another bolt is not allowed. It means nut can only be compared with bolt and bolt can only be compared with nut to see which one is bigger/smaller.

Examples:

```
Input : nuts[] = {'@', '#', '$', '%', '^', '&'}
        bolts[] = {'$', '%', '&', '^', '@', '#'}
Output : Matched nuts and bolts are-
        $ % & ^ @ #
        $ % & ^ @ #
```

Other way of asking this problem is, given a box with locks and keys where one lock can be opened by one key in the box. We need to match the pair.

We have discussed a sorting based solution in below post.

Nuts & Bolts Problem (Lock & Key problem) | Set 1

In this post, hashmap based approach is discussed.

1. Travese the nuts array and create a hashmap
2. Traverse the bolts array and search for it in hashmap.

3. If it is found in the hashmap of nuts than this means bolts exist for that nut.

```
 // Hashmap based solution to solve
#include <bits/stdc++.h>
using namespace std;

// function to match nuts and bolts
void nutboltmatch(char nuts[], char bolts[], int n)
{
    unordered_map<char, int> hash;

    // creating a hashmap for nuts
    for (int i = 0; i < n; i++)
        hash[nuts[i]] = i;

    // searching for nuts for each bolt in hash map
    for (int i = 0; i < n; i++)
        if (hash.find(bolts[i]) != hash.end())
            nuts[i] = bolts[i];

    // print the result
    cout << "matched nuts and bolts are-" << endl;
    for (int i = 0; i < n; i++)
        cout << nuts[i] << " ";
    cout << endl;
    for (int i = 0; i < n; i++)
        cout << bolts[i] << " ";
}

// Driver code
int main()
{
    char nuts[] = {'@', '#', '$', '%', '^', '&'};
    char bolts[] = {'$', '%', '&', '^', '@', '#'};
    int n = sizeof(nuts) / sizeof(nuts[0]);
    nutboltmatch(nuts, bolts, n);
    return 0;
}
```

Output:

```
matched nuts and bolts are-
$ % & ^ @ #
$ % & ^ @ #
```

Time complexity for this solution is O(n).

Source

https://www.geeksforgeeks.org/nuts-bolts-problem-lock-key-problem-set-2-hashmap/

Chapter 232

Only integer with positive value in positive negative value in array

Only integer with positive value in positive negative value in array - GeeksforGeeks

Given an array of **N** integers. In the given, for each positive element 'x' there exist a negative value '-x', except one integer whose negative value is not present. That integer may occur multiple number of time. The task is find that integer.

Examples:

```
Input : arr[] = { 1, 8, -6, -1, 6, 8, 8 }
Output : 8
All positive elements have an equal negative
value except 8.

Input : arr[] = { 15, 6, -9, 4, 15, 9,
                      -6, -4, 15, 15 }
Output : 15
```

Method 1: (hashing) The idea is to create a hash table, initialize with zero value. Whenever we encounter a positive value, we add +1 on corresponding index position in hash. And whenever we encounter negative value we add -1. Finally we traverse the whole hash. After traversing, the index with non-zero value is the only integer with only value without negative pair.

Below is C++ implementation of this approach:

```
 // A hashing based solution to find the only
// element that doesn't have negative value.
```

```
#include <bits/stdc++.h>
using namespace std;

// Return the integer with have no negative value.
int findInteger(int arr[], int n)
{
    unorder_map<int, int> hash;
    int maximum = 0;

    // Traversing the array.
    for (int i = 0; i < n; i++) {

        // If negative, then subtract 1 in hash array.
        if (arr[i] < 0)
            hash[abs(arr[i])] -= 1;

        // Else add 1 in hash array.
        else
            hash[arr[i]] += 1;
    }

    // Traverse the hash array.
    for (int i = 0; i < n; i++)
        if (hash[arr[i]] == 0)
            return i;

    return -1;
}

// Driven Program
int main()
{
    int arr[] = { 1, 8, -6, -1, 6, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findInteger(arr, n) << endl;
    return 0;
}
```

Output:

```
8
```

Method 2: (efficient) The idea is to find the sum of each element of array and also count the number of even and odd numbers in the array. Finally, divide the sum by absolute difference of number of odd element and even element.

Below is the implementation of this approach:

C++

```
 // An efficient solution to find the only
// element that doesn't have negative value.
#include <bits/stdc++.h>
using namespace std;

// Return the integer with have no negative value.
int findInteger(int arr[], int n)
{
    int neg = 0, pos = 0;
    int sum = 0;

    for (int i = 0; i < n; i++) {
        sum += arr[i];

        // If negative, then increment neg count.
        if (arr[i] < 0)
            neg++;

        // Else increment pos count..
        else
            pos++;
    }

    return (sum / abs(neg - pos));
}

// Driven Program
int main()
{
    int arr[] = { 1, 8, -6, -1, 6, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findInteger(arr, n) << endl;
    return 0;
}
```

Java

```
 // An efficient solution to find the
// only element that doesn't have
// negative value.
import java.lang.*;

class GFG {
```

```
    // Return the integer with have
    // no negative value.
    static int findInteger(int arr[], int n)
    {
        int neg = 0, pos = 0;
        int sum = 0;

        for (int i = 0; i < n; i++)
        {
            sum += arr[i];

            // If negative, then increment
            // neg count.
            if (arr[i] < 0)
                neg++;

            // Else increment pos count..
            else
                pos++;
        }

        return (sum / Math.abs(neg - pos));
    }

    // Driven Program
    public static void main(String[] args)
    {

        int arr[] = new int[]{ 1, 8, -6, -1, 6, 8 };
        int n = arr.length;

        System.out.println(findInteger(arr, n));
    }
}

// This code is contributed by Smitha.
```

Python 3

```
 # An efficient solution to find the
# only element that doesn't have
# negative value.

# Return the integer with have no
# negative value.
def findInteger(arr, n):
```

```
    neg = 0
    pos = 0
    sum = 0

    for i in range(0, n):
        sum += arr[i]

        # If negative, then increment
        # neg count.
        if (arr[i] < 0):
            neg += 1

        # Else increment pos count..
        else:
            pos += 1

    return (sum / abs(neg - pos))

# Driven Program
arr = [1, 8, -6, -1, 6, 8 ]
n = len(arr)
print(int(findInteger(arr, n)))

# This code is contributed by Smitha.
```

C#

```
 // An efficient solution to find the
// only element that doesn't have
// negative value.
using System;

class GFG {

    // Return the integer with have
    // no negative value.
    static int findInteger(int [] arr, int n)
    {
        int neg = 0, pos = 0;
        int sum = 0;

        for (int i = 0; i < n; i++)
        {
            sum += arr[i];

            // If negative, then increment
            // neg count.
            if (arr[i] < 0)
```

```
                neg++;

            // Else increment pos count..
            else
                pos++;
        }

        return (sum / Math.Abs(neg - pos));
    }

    // Driven Program
    public static void Main()
    {
        int [] arr = new int[]{ 1, 8, -6,
                                -1, 6, 8 };
        int n = arr.Length;

        Console.Write(findInteger(arr, n));
    }
}

// This code is contributed by Smitha.
```

PHP

```
 <?php
// An efficient solution to find the only
// element that doesn't have negative value.

// Return the integer with
// have no negative value.
function findInteger($arr, $n)
{
    $neg = 0; $pos = 0;
    $sum = 0;

    for ($i = 0; $i < $n; $i++)
    {
        $sum += $arr[$i];

        // If negative, then
        // increment neg count.
        if ($arr[$i] < 0)
            $neg++;

        // Else increment
        // pos count..
        else
```

```
            $pos++;
    }

    return ($sum / abs($neg - $pos));
}

    // Driver Code
    $arr = array(1, 8, -6, -1, 6, 8);
    $n = sizeof($arr);
    echo findInteger($arr, $n) ,"\n";

// This code is contributed by aj_36
?>
```

Output:

```
8
```

Improved By : Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/integer-positive-value-positive-negative-value-array/

Chapter 233

Overview of Data Structures | Set 2 (Binary Tree, BST, Heap and Hash)

Overview of Data Structures | Set 2 (Binary Tree, BST, Heap and Hash) - GeeksforGeeks

We have discussed Overview of Array, Linked List, Queue and Stack. In this article following Data Structures are discussed.

5. Binary Tree
6. Binary Search Tree
7. Binary Heap
9. Hashing

Binary Tree
Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.
A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. It is implemented mainly using Links.

Binary Tree Representation: A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL. A Binary Tree node contains following parts.
1. Data
2. Pointer to left child
3. Pointer to right child

A Binary Tree can be traversed in two ways:
Depth First Traversal: Inorder (Left-Root-Right), Preorder (Root-Left-Right) and Postorder (Left-Right-Root)
Breadth First Traversal: Level Order Traversal

Binary Tree Properties:

```
The maximum number of nodes at level 'l' = 2l-1.

Maximum number of nodes = 2h - 1.
Here h is height of a tree. Height is considered
as is maximum number of nodes on root to leaf path

Minimum possible height =  ceil(Log2(n+1))

In Binary tree, number of leaf nodes is always one
more than nodes with two children.

Time Complexity of Tree Traversal: O(n)
```

Examples : One reason to use binary tree or tree in general is for the things that form a hierarchy. They are useful in File structures where each file is located in a particular directory and there is a specific hierarchy associated with files and directories. Another example where Trees are useful is storing heirarchical objects like JavaScript Document Object Model considers HTML page as a tree with nesting of tags as parent child relations.

Binary Search Tree
In Binary Search Tree is a Binary Tree with following additional properties:
1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. The left and right subtree each must also be a binary search tree.

Time Complexities:

```
Search :  O(h)
Insertion : O(h)
Deletion : O(h)
Extra Space : O(n) for pointers

h: Height of BST
n: Number of nodes in BST

If Binary Search Tree is Height Balanced,
then h = O(Log n)

Self-Balancing BSTs such as AVL Tree, Red-Black
Tree and Splay Tree make sure that height of BST
remains O(Log n)
```

BST provide moderate access/search (quicker than Linked List and slower than arrays).
BST provide moderate insertion/deletion (quicker than Arrays and slower than Linked Lists).

Examples : Its main use is in search application where data is constantly entering/leaving and data needs to printed in sorted order. For example in implementation in E- commerce

websites where a new product is added or product goes out of stock and all products are lised in sorted order.

Binary Heap
A Binary Heap is a Binary Tree with following properties.
1) It's a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.
2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to Min Heap. It is mainly implemented using array.

```
Get Minimum in Min Heap: O(1) [Or Get Max in Max Heap]
Extract Minimum Min Heap: O(Log n) [Or Extract Max in Max Heap]
Decrease Key in Min Heap: O(Log n)  [Or Extract Max in Max Heap]
Insert: O(Log n)
Delete: O(Log n)
```

Example : Used in implementing efficient priority-queues, which in turn are used for scheduling processes in operating systems. Priority Queues are also used in Dijstra's and Prim's graph algorithms.
The Heap data structure can be used to efficiently find the k smallest (or largest) elements in an array, merging k sorted arrays, median of a stream, etc.
Heap is a special data structure and it cannot be used for searching of a particular element.

HashingHash Function: A function that converts a given big input key to a small practical integer value. The mapped integer value is used as an index in hash table. A good hash function should have following properties
1) Efficiently computable.
2) Should uniformly distribute the keys (Each table position equally likely for each key)

Hash Table: An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.

Collision Handling: Since a hash function gets us a small number for a key which is a big integer or string, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

Chaining:The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.
Open Addressing: In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table.

```
Space : O(n)
Search    : O(1) [Average]    O(n) [Worst case]
Insertion : O(1) [Average]    O(n) [Worst Case]
Deletion  : O(1) [Average]    O(n) [Worst Case]
```

Hashing seems better than BST for all the operations. But in hashing, elements are unordered and in BST elements are stored in an ordered manner. Also BST is easy to implement but hash functions can sometimes be very complex to generate. In BST, we can also efficiently find floor and ceil of values.

Example : Hashing can be used to remove duplicates from a set of elements. Can also be used find frequency of all items. For example, in web browsers, we can check visited urls using hashing. In firewalls, we can use hashing to detect spam. We need to hash IP addresses. Hashing can be used in any situation where want search() insert() and delete() in O(1) time.

This article is contributed by **Abhiraj Smit**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Rahul1421

Source

https://www.geeksforgeeks.org/overview-of-data-structures-set-2-binary-tree-bst-heap-and-hash/

Chapter 234

Pair with given product | Set 1 (Find if any pair exists)

Pair with given product | Set 1 (Find if any pair exists) - GeeksforGeeks

Given an array of distinct elements and a number x, find if there is a pair with product equal to x.

Examples :

```
Input : arr[] = {10, 20, 9, 40};
        int x = 400;
Output : Yes

Input : arr[] = {10, 20, 9, 40};
        int x = 190;
Output : No

Input : arr[] = {-10, 20, 9, -40};
        int x = 400;
Output : Yes

Input : arr[] = {-10, 20, 9, 40};
        int x = -400;
Output : Yes

Input : arr[] = {0, 20, 9, 40};
        int x = 0;
Output : Yes
```

Naive approach ($O(n^2)$) is to run two loops to consider all possible pairs. For every pair, check if product is equal to x or not.

C++

```
 // A simple C++ program to find if there is a pair
// with given product.
#include<bits/stdc++.h>
using namespace std;

// Returns true if there is a pair in arr[0..n-1]
// with product equal to x.
bool isProduct(int arr[], int n, int x)
{
    // Consider all possible pairs and check for
    // every pair.
    for (int i=0; i<n-1; i++)
       for (int j=i+1; i<n; i++)
          if (arr[i] * arr[j] == x)
              return true;

    return false;
}

// Driver code
int main()
{
    int arr[] = {10, 20, 9, 40};
    int x = 400;
    int n = sizeof(arr)/sizeof(arr[0]);
    isProduct(arr, n, x)? cout << "Yesn"
                        : cout << "Non";
    x = 190;
    isProduct(arr, n, x)? cout << "Yesn"
                        : cout << "Non";
    return 0;
}
```

Java

```
 // Java program to find if there is a pair
// with given product.
class GFG
{
    // Returns true if there is a pair in
    // arr[0..n-1] with product equal to x.
    boolean isProduct(int arr[], int n, int x)
    {
        for (int i=0; i<n-1; i++)
            for (int j=i+1; j<n; j++)
```

```
                if (arr[i]*arr[j] == x)
                    return true;
        return false;
    }

    // Driver code
    public static void main(String[] args)
    {
        GFG g = new GFG();
        int arr[] = {10, 20, 9, 40};
        int x = 400;
        int n = arr.length;
        if (g.isProduct(arr, n, x))
            System.out.println("Yes");
        else
            System.out.println("No");

        x = 190;
        if (g.isProduct(arr, n, x))
            System.out.println("Yes");
        else
            System.out.println("No");

    }
}
// This code is contributed by Kamal Rawal
```

Python3

```
 # Python3 program to find if there
# is a pair with given product.

# Returns true if there is a
# pair in arr[0..n-1] with
# product equal to x
def isProduct(arr, n, x):
    for i in arr:
        for j in arr:
            if i * j == x:
                return True
    return False


# Driver code
arr = [10, 20, 9, 40]
x = 400
n = len(arr)
if(isProduct(arr,n, x) == True):
```

```
    print ("Yes")

else:
    print("No")

x = 900
if(isProduct(arr, n, x)):
    print("Yes")

else:
    print("No")

# This code is contributed
# by prerna saini
```

C#

```
 // C# program to find
// if there is a pair
// with given product.
using System;

class GFG
{

// Returns true if there
// is a pair in arr[0..n-1]
// with product equal to x.
static bool isProduct(int []arr,
                      int n, int x)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] * arr[j] == x)
                return true;
    return false;
}

// Driver Code
static void Main()
{
    int []arr = {10, 20, 9, 40};
    int x = 400;
    int n = arr.Length;
    if (isProduct(arr, n, x))
        Console.WriteLine("Yes");
    else
```

```
        Console.WriteLine("No");

    x = 190;
    if (isProduct(arr, n, x))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed
// by Sam007
```

PHP

```
 <?php
// A simple php program to
// find if there is a pair
// with given product.

// Returns true if there
// is a pair in arr[0..n-1]
// with product equal to x.
function isProduct($arr, $n, $x)
{
    // Consider all possible
    // pairs and check for
    // every pair.
    for ($i = 0;
         $i < $n - 1; $i++)
    for ($j = $i + 1;
         $i < $n; $i++)
        if ($arr[$i] *
            $arr[$j] == $x)
            return true;

    return false;
}

// Driver code
$arr = array(10, 20, 9, 40);
$x = 400;
$n = count($arr);
if(isProduct($arr, $n, $x))
echo "Yes\n";
else
echo "No\n";
```

```
$x = 190;
if(isProduct($arr, $n, $x))
echo "Yes\n";
else
echo "No\n";

// This code is contributed
// by Sam007
?>
```

Output :

```
Yes
No
```

Better Solution (O(n Log n) : We sort the given array. After sorting, we traverse the array and for every element arr[i], we do binary search for x/arr[i] in the subarry on right of arr[i], i.e., in subarray arr[i+1..n-1]

Efficient Solution (O(n)): We can improve time complexity to O(n) using hashing. Below are steps.

1. Create an empty hash table
2. Traverse array elments and do following for every element arr[i].
   - If arr[i] is 0 and x is also 0, return true, else ignore arr[i].
   - If x % arr[i] is 0 and x/arr[i] exists in table, return true.
   - Insert arr[i] into the hash table.
3. Return false

Below is the implementation of above idea.

C++

```
 // C++ program to find if there is a pair
// with given product.
#include<bits/stdc++.h>
using namespace std;

// Returns true if there is a pair in arr[0..n-1]
// with product equal to x.
```

```
bool isProduct(int arr[], int n, int x)
{
    if (n < 2)
        return false;

    // Create an empty set and insert first
    // element into it
    unordered_set<int> s;

    // Traverse remaining elements
    for (int i=0; i<n; i++)
    {
        // 0 case must be handles explicitly as
        // x % 0 is undefined behaviour in C++
        if (arr[i] == 0)
        {
           if (x == 0)
               return true;
           else
               continue;
        }

        // x/arr[i] exists in hash, then we
        // found a pair
        if (x%arr[i] == 0)
        {
            if (s.find(x/arr[i]) != s.end())
               return true;

            // Insert arr[i]
            s.insert(arr[i]);
        }
    }
    return false;
}

// Driver code
int main()
{
    int arr[] = {10, 20, 9, 40};
    int x = 400;

    int n = sizeof(arr)/sizeof(arr[0]);
    isProduct(arr, n, x)? cout << "Yes\n"
                        : cout << "Non";

    x = 190;
    isProduct(arr, n, x)? cout << "Yes\n"
```

```
                        : cout << "Non";

    return 0;
}
```

Java

```
 // Java program if there exists a pair for given product
import java.util.HashSet;

class GFG
{
    // Returns true if there is a pair in arr[0..n-1]
    // with product equal to x.
    static boolean isProduct(int arr[], int n, int x)
    {
        // Create an empty set and insert first
        // element into it
        HashSet<Integer> hset = new HashSet<>();

        if(n < 2)
            return false;

        // Traverse remaining elements
        for(int i = 0; i < n; i++)
        {
            // 0 case must be handles explicitly as
            // x % 0 is undefined
            if(arr[i] == 0)
            {
                if(x == 0)
                    return true;
                else
                    continue;
            }

            // x/arr[i] exists in hash, then we
            // found a pair
            if(x % arr[i] == 0)
            {
                if(hset.contains(x / arr[i]))
                    return true;

            // Insert arr[i]
            hset.add(arr[i]);
            }
        }
        return false;
```

```
    }

    // driver code
    public static void main(String[] args)
    {
        int arr[] = {10, 20, 9, 40};
        int x = 400;
        int n = arr.length;

        if(isProduct(arr, arr.length, x))
        System.out.println("Yes");
        else
        System.out.println("No");

        x = 190;

        if(isProduct(arr, arr.length, x))
        System.out.println("Yes");
        else
        System.out.println("No");
    }
}

// This code is contributed by Kamal Rawal
```

Output :

```
Yes
No
```

In the next set, we will be discussing approach to print all pairs with product equal to 0.

This article is contributed by **Shubham Goyal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007

Source

https://www.geeksforgeeks.org/pair-with-given-product-set-1-find-if-any-pair-exists/

Chapter 235

Pair with given sum and maximum shortest distance from end

Pair with given sum and maximum shortest distance from end - GeeksforGeeks

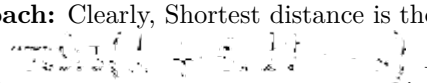
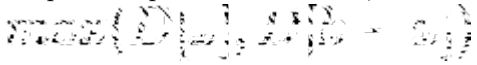
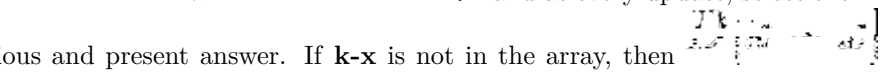
Given an array of N integers and an integer K, pick two distinct elements whose sum is K and find the maximum shortest distance of the picked elements from the endpoints.

Examples:

```
Input : a[] = {2, 4, 3, 2, 1}
        k = 5.
Output :  2
Explanation:
Select the pair(4, 1).
Shortest distance of 4 from ends = 2
Shortest distance of 1 from ends = 1
Hence, answer is max(2, 1) = 2

Input : a[] = {2, 4, 1, 9, 5}
        k = 3
Output : 3
Explanation:
Select the pair (2, 1)
Shortest distance of 2 from ends = 1
Shortest distance of 1 from ends = 3
Hence, answer is max(1, 3) = 3.
```

Note: The distance of end elements from ends is 1 and not 0.

Naive approach: The approach is to run two loops and in inner loop check if two elements are making a pair with sum k. If yes, then make answer as maximum of the shortest distances of two elements, compare it with the previous pair's answer and make answer as minimum of these two. When the loop ends we get the desired output.

Efficient Approach: Clearly, Shortest distance is the distance from left end and distance from right end i.e,  . Let us denote shortest distance of i-th element as . There is another case where an element in the selected pair is repeated then select minimum of all the shortest distances of occurrences of that element. Run a loop and store shortest distance of all the array elements in another array(let it be ). Now, we got shortest distances of all the elements.
Run a for loop. If the picked element is **x**, then the other element should be **k-x**. Update the ans with  and at every update, select the minimum of previous and present answer. If **k-x** is not in the array, then  will be INFINITE, which will be initialized already.

```
 // C++ code to find maximum shortest distance
// from endpoints
#include <bits/stdc++.h>
using namespace std;

// function to find maximum shortest distance
int find_maximum(int a[], int n, int k)
{
    // stores the shortest distance of every
    // element in original array.
    unordered_map<int, int> b;

    for (int i = 0; i < n; i++) {
        int x = a[i];

        // shortest distance from ends
        int d = min(1 + i, n - i);
        if (b.find(x) == b.end())
            b[x] = d;

        else

            /* if duplicates are found, b[x]
            is replaced with minimum of the
            previous and current position's
            shortest distance*/
            b[x] = min(d, b[x]);
    }
```

```
    int ans = INT_MAX;
    for (int i = 0; i < n; i++) {
        int x = a[i];

        // similar elements ignore them
        // cause we need distinct elements
        if (x != k - x && b.find(k - x) != b.end())
            ans = min(max(b[x], b[k - x]), ans);
    }
    return ans;
}

// driver code
int main()
{
    int a[] = { 3, 5, 8, 6, 7 };
    int K = 11;
    int n = sizeof(a) / sizeof(a[0]);
    cout << find_maximum(a, n, K) << endl;
    return 0;
}
```

Output:

```
 2
```

Source

https://www.geeksforgeeks.org/print-maximum-shortest-distance/

Chapter 236

Pairs of Amicable Numbers

Pairs of Amicable Numbers - GeeksforGeeks

Given an array of integers, print the number of pairs in the array that form an amicable pair. Two numbers are amicable if the first is equal to the sum of divisors of the second, and if the second number is equal to the sum of divisors of the first.

Examples :

```
Input  : arr[] = {220, 284, 1184, 1210, 2, 5}
Output : 2
Explanation : (220, 284) and (1184, 1210)
              form amicable pair

Input  : arr[] = {2620, 2924, 5020, 5564, 6232, 6368}
Output : 3
Explanation : (2620, 2924), (5020, 5564) and (6232, 6368)
              forms amicable pair
```

A **simple solution** is to traverse each pair and check if they form an amicable pair, if they do we increment the count.

C++

```
 // A simple C++ program to count
// amicable pairs in an array.
#include <bits/stdc++.h>
using namespace std;

// Calculate the sum
// of proper divisors
int sumOfDiv(int x)
```

```
{
    // 1 is a proper divisor
    int sum = 1;
    for (int i = 2; i <= sqrt(x); i++)
    {
        if (x % i == 0)
        {
            sum += i;

            // To handle perfect squares
            if (x / i != i)
                sum += x / i;
        }
    }
    return sum;
}

// Check if pair is amicable
bool isAmicable(int a, int b)
{
    return (sumOfDiv(a) == b &&
            sumOfDiv(b) == a);
}

// This function prints pair
// of amicable pairs present
// in the input array
int countPairs(int arr[], int n)
{
    int count = 0;

    // Iterate through each
    // pair, and find if it
    // an amicable pair
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (isAmicable(arr[i], arr[j]))
                count++;

    return count;
}

// Driver code
int main()
{
    int arr1[] = { 220, 284, 1184,
                   1210, 2, 5 };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
```

```
    cout << countPairs(arr1, n1)
         << endl;

    int arr2[] = { 2620, 2924, 5020,
                   5564, 6232, 6368 };
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    cout << countPairs(arr2, n2);
    return 0;
}
```

Java

```
 // A simple Java program to count
// amicable pairs in an array.
import java.io.*;

class GFG
{

    // Calculate the sum
    // of proper divisors
    static int sumOfDiv(int x)
    {

        // 1 is a proper divisor
        int sum = 1;
        for (int i = 2; i <= Math.sqrt(x); i++)
        {
            if (x % i == 0)
            {
                sum += i;

                // To handle perfect squares
                if (x / i != i)
                    sum += x / i;
            }
        }

        return sum;
    }

    // Check if pair is amicable
    static boolean isAmicable(int a, int b)
    {
        return (sumOfDiv(a) == b &&
                sumOfDiv(b) == a);
    }
```

```
    // This function prints pair
    //  of amicable pairs present
    // in the input array
    static int countPairs(int arr[], int n)
    {
        int count = 0;

        // Iterate through each pair,
        // and find if it an amicable pair
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (isAmicable(arr[i], arr[j]))
                    count++;

        return count;
    }

    // Driver code
    public static void main(String args[])
    {

        int arr1[] = { 220, 284, 1184,
                       1210, 2, 5 };
        int n1 = arr1.length;

        System.out.println(countPairs(arr1, n1));

        int arr2[] = { 2620, 2924, 5020,
                       5564, 6232, 6368 };
        int n2 = arr2.length;

        System.out.println(countPairs(arr2, n2));
    }
}

// This code is contributed by Anshika Goyal.
```

Python3

```
 # Python3 program to count
# amicable pairs in an array

# Calculate the sum
# of proper divisors
def sumOfDiv(x):
    sum = 1
    for i in range(2, x):
        if x % i == 0:
```

```
            sum += i
    return sum

# Check if pair is amicable
def isAmicable(a, b):
    if sumOfDiv(a) == b and sumOfDiv(b) == a:
        return True
    else:
        return False

# This function prints pair
# of amicable pairs present
# in the input array
def countPairs(arr, n):
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            if isAmicable(arr[i], arr[j]):
                count = count + 1
    return count

# Driver Code
arr1 = [220, 284, 1184,
        1210, 2, 5]
n1 = len(arr1)
print(countPairs(arr1, n1))

arr2 = [2620, 2924, 5020,
        5564, 6232, 6368]
n2 = len(arr2)
print(countPairs(arr2, n2))

# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // A simple C# program to count
// amicable pairs in an array.
using System;

class GFG
{

    // Calculate the sum
    // of proper divisors
    static int sumOfDiv(int x)
    {
```

```
    // 1 is a proper divisor
    int sum = 1;
    for (int i = 2; i <= Math.Sqrt(x); i++)
    {
        if (x % i == 0)
        {
            sum += i;

            // To handle perfect squares
            if (x / i != i)
                sum += x / i;
        }
    }

    return sum;
}

// Check if pair is amicable
static bool isAmicable(int a, int b)
{
    return (sumOfDiv(a) == b &&
            sumOfDiv(b) == a);
}

// This function prints pair
// of amicable pairs present
// in the input array
static int countPairs(int []arr, int n)
{
    int count = 0;

    // Iterate through each pair,
    // and find if it an amicable pair
    for (int i = 0; i < n; i++)

        for (int j = i + 1; j < n; j++)
            if (isAmicable(arr[i], arr[j]))
                count++;

    return count;
}

// Driver code
public static void Main()
{

    int []arr1 = {220, 284, 1184,
```

```
                    1210, 2, 5};
        int n1 = arr1.Length;

        Console.WriteLine(countPairs(arr1, n1));

        int []arr2 = {2620, 2924, 5020,
                    5564, 6232, 6368};
        int n2 = arr2.Length;

        Console.WriteLine(countPairs(arr2, n2));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// A simple PHP program to count
// amicable pairs in an array.

// Calculate the sum
// of proper divisors
function sumOfDiv( $x)
{
    // 1 is a proper divisor
    $sum = 1;
    for ( $i = 2; $i <= sqrt($x); $i++)
    {
        if ($x % $i == 0)
        {
            $sum += $i;

            // To handle perfect squares
            if ($x / $i != $i)
                $sum += $x / $i;
        }
    }
    return $sum;
}

// Check if pair is amicable
function isAmicable( $a, $b)
{
    return (sumOfDiv($a) == $b and
            sumOfDiv($b) == $a);
}
```

```
// This function prints pair
// of amicable pairs present
// in the input array
function countPairs( $arr, $n)
{
    $count = 0;

    // Iterate through each pair,
    // and find if it an amicable pair
    for ( $i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)
            if (isAmicable($arr[$i], $arr[$j]))
                $count++;

    return $count;
}

// Driver code
$arr1 = array( 220, 284, 1184,
               1210, 2, 5 );
$n1 = count($arr1);
echo countPairs($arr1, $n1),"\n";

$arr2 = array( 2620, 2924, 5020,
               5564, 6232, 6368 );
$n2 = count($arr2);
echo countPairs($arr2, $n2);

// This code is contributed by anuj_67.
?>
```

Output :

```
2
3
```

An **efficient solution** is be to keep the numbers stored in a map and for every number we find the sum of its proper divisor and check if that's also present in the array. If it is present, we can check if they form an amicable pair or not.

Thus, the complexity would be considerably reduced. Below is the C++ program for the same.

```
 // Efficient C++ program to count
// Amicable pairs in an array.
#include <bits/stdc++.h>
using namespace std;
```

```
// Calculate the sum
// of proper divisors
int sumOfDiv(int x)
{
    // 1 is a proper divisor
    int sum = 1;
    for (int i = 2; i <= sqrt(x); i++)
    {
        if (x % i == 0) {
            sum += i;

            // To handle perfect squares
            if (x / i != i)
                sum += x / i;
        }
    }
    return sum;
}

// Check if pair is amicable
bool isAmicable(int a, int b)
{
    return (sumOfDiv(a) == b &&
            sumOfDiv(b) == a);
}

// This function prints count
// of amicable pairs present
// in the input array
int countPairs(int arr[], int n)
{
    // Map to store the numbers
    unordered_set<int> s;
    int count = 0;
    for (int i = 0; i < n; i++)
        s.insert(arr[i]);

    // Iterate through each number,
    // and find the sum of proper
    // divisors and check if it's
    // also present in the array
    for (int i = 0; i < n; i++)
    {
        if (s.find(sumOfDiv(arr[i])) != s.end())
        {
            // It's sum of proper divisors
            int sum = sumOfDiv(arr[i]);
```

```
            if (isAmicable(arr[i], sum))
                count++;
        }
    }

    // As the pairs are counted
    // twice, thus divide by 2
    return count / 2;
}

// Driver code
int main()
{
    int arr1[] = { 220, 284, 1184,
                   1210, 2, 5 };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    cout << countPairs(arr1, n1)
         << endl;

    int arr2[] = { 2620, 2924, 5020,
                   5564, 6232, 6368 };
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    cout << countPairs(arr2, n2)
         << endl;
    return 0;
}
```

Output :

```
2
3
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/pairs-amicable-numbers/

Chapter 237

Pairs of Positive Negative values in an array

Pairs of Positive Negative values in an array - GeeksforGeeks

Given an array of **distinct** integers, print all the pairs having positive value and negative value of a number that exists in the array. We need to print pairs in order of their occurrences. A pair whose any element appears first should be printed first.

Examples:

```
Input  :  arr[] = { 1, -3, 2, 3, 6, -1 }
Output : -1 1 -3 3

Input  :  arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 }
Output : -1 1 -4 4 -8 8 -9 9
```

Method 1 (Simple : $O(n^2)$)
The idea is to use two nested loop. For each element arr[i], find negative of arr[i] from index i + 1 to n – 1 and store it in another array. For output, sort the stored element and print negative positive value of the stored element.

Below is the implementation of this approach:

C++

```
 // Simple CPP program to find pairs of positive
// and negative values present in an array.
#include <bits/stdc++.h>
using namespace std;

// Print pair with negative and positive value
```

```
void printPairs(int arr[], int n)
{
    vector<int> v;

    // For each element of array.
    for (int i = 0; i < n; i++)

        // Try to find the negative value of
        // arr[i] from i + 1 to n
        for (int j = i + 1; j < n; j++)

            // If absolute values are equal print pair.
            if (abs(arr[i]) == abs(arr[j]))
                v.push_back(abs(arr[i]));

    // If size of vector is 0, therefore there is no
    // element with positive negative value, print "0"
    if (v.size() == 0)
       return;

    // Sort the vector
    sort(v.begin(), v.end());

    // Print the pair with negative positive value.
    for (int i = 0; i < v.size(); i++)
        cout << -v[i] << " " << v[i];
}

// Driven Program
int main()
{
    int arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printPairs(arr, n);
    return 0;
}
```

Java

```
 // Java program to find pairs of positive
// and negative values present in an array.
import java.util.*;
import java.lang.*;

class GFG {

    // Print pair with negative and positive value
    public static void printPairs(int arr[] , int n)
```

```
    {
        Vector<Integer> v = new Vector<Integer>();
        // For each element of array.
        for (int i = 0; i < n; i++)

            // Try to find the negative value of
            // arr[i] from i + 1 to n
            for (int j = i + 1; j < n; j++)

                // If absolute values are equal
                // print pair.
                if (Math.abs(arr[i]) ==
                                    Math.abs(arr[j]))
                    v.add(Math.abs(arr[i]));


        // If size of vector is 0, therefore there
        // is no element with positive negative
        // value, print "0"
        if (v.size() == 0)
            return;

        // Sort the vector
        Collections.sort(v);

        // Print the pair with negative positive
        // value.
        for (int i = 0; i < v.size(); i++)
            System.out.print(-v.get(i) + " " +
                                    v.get(i));
    }

    // Driven Program
    public static void main(String[] args)
    {
        int arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 };
        int n = arr.length;
        printPairs(arr, n);
    }
}

// This code is contributed by Prasad Kshirsagar.
```

Output:

```
-1 1-4 4-8 8-9 9
```

Method 2 (Hahing)
The idea is to use hashing. Traverse the given array, increase the count at absolute value of hash table. If count becomes 2, store its absolute value in another vector. And finally sort the vector. If the size of the vector is 0, print "0", else for each term in vector print first its negative value and the the positive value.

Below is C++ implementation of this approach:

```
 // Efficient CPP program to find pairs of
// positive and negative values present in
// an array.
#include <bits/stdc++.h>
using namespace std;

// Print pair with negative and positive value
void printPairs(int arr[], int n)
{
    vector<int> v;
    unordered_map<int, bool> cnt;

    // For each element of array.
    for (int i = 0; i < n; i++) {

        // If element has not encounter early,
        // mark it on cnt array.
        if (cnt[abs(arr[i])] == 0)
            cnt[abs(arr[i])] = 1;

        // If seen before, push it in vector (
        // given that elements are distinct)
        else {
            v.push_back(abs(arr[i]));
            cnt[abs(arr[i])] = 0;
        }
    }

    if (v.size() == 0)
        return;

    sort(v.begin(), v.end());
    for (int i = 0; i < v.size(); i++)
        cout << -v[i] << " " << v[i] << " ";
}

// Driven Program
int main()
{
    int arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 };
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    printPairs(arr, n);
    return 0;
}
```

Output:

```
-1 1 -4 4 -8 8 -9 9
```

Improved By : Prasad_Kshirsagar

Source

https://www.geeksforgeeks.org/pairs-of-positive-negative-values-in-an-array/

Chapter 238

Palindrome Substring Queries

Palindrome Substring Queries - GeeksforGeeks

Given a string and several queries on the substrings of the given input string to check whether the substring is a palindrome or not.

Examples :
Suppose our input string is "abaaabaaaba" and the queries- [0, 10], [5, 8], [2, 5], [5, 9]

We have to tell that the substring having the starting and ending indices as above is a palindrome or not.

[0, 10] → Substring is "abaaabaaaba" which is a palindrome.
[5, 8] → Substring is "baaa" which is not a palindrome.
[2, 5] → Substring is "aaab" which is not a palindrome.
[5, 9] → Substring is "baaab" which is a palindrome.

Let us assume that there are Q such queries to be answered and N be the length of our input string. There are the following two ways to answer these queries

Method 1 (Naive)

One by one we go through all the substrings of the queries and check whether the substring under consideration is a palindrome or not.

Since there are Q queries and each query can take O(N) worse case time to answer, this method takes O(Q.N) time in the worst case. Although this is an in-place/space-efficient algorithm, but still there are more efficient method to do this.

Method 2 (Cumulative Hash)

The idea is similar to Rabin Karp string matching. We use string hashing. What we do is that we calculate cumulative hash values of the string in the original string as well as the reversed string in two arrays- prefix[] and suffix[].

How to calculate the cumulative hash values ?

Suppose our string is str[], then the cumulative hash function to fill our prefix[] array used is-

```
prefix[0] = 0
prefix[i] = str[0] + str[1] * 101 + str[2] * 1012 +
                              ..... + str[i-1] * 101i-1

For example, take the string- "abaaabxyaba"

prefix[0] = 0
prefix[1] = 97  (ASCII Value of 'a' is 97)
prefix[2] = 97 + 98 * 101
prefix[3] = 97 + 98 * 101 + 97 * 1012
.........................
.........................
prefix[11] = 97 + 98 * 101 + 97 * 1012 +
                        ........+ 97 * 10110
```

Now the reason to store in that way is that we can easily find the hash value of any substring in O(1) time using-

```
 hash(L, R) = prefix[R+1] - prefix[L]
```

For example, hash (1, 5) = hash ("baaab") = prefix[6] – prefix[1] = 98 * 101 + 97 * 1012 + 97 * 1013 + 97 * 1014 + 98 * 1015 = 1040184646587 [We will use this weird value later to explain what's happening].

Similar to this we will fill our suffix[] array as-

```
suffix[0] = 0
suffix[i] = str[n-1] + str[n-2] * 101 + str[n-3] * 1012 +
                                  ..... + str[n-i] * 101i-1

For example, take the string- "abaaabxyaba"

suffix[0] = 0
suffix[1] = 97  (ASCII Value of 'a' is 97)
suffix[2] = 97 + 98 * 101
suffix[3] = 97 + 98 * 101 + 97 * 1012
.........................
.........................
suffix[11] = 97 + 98 * 101 + 97 * 1012 + ........+ 97 * 10110
```

Now the reason to store in that way is that we can easily find the reverse hash value of any substring in O(1) time using

```
reverse_hash(L, R) = hash (R, L) = suffix[n-L] - suffix[n-R-1]
```

where n = length of string.

For “abaaabxyaba”, n = 11
reverse_hash(1,5) = reverse_hash(“baaab”) = hash(“baaab”) [Reversing “baaab” gives “baaab”]

hash(“baaab”) = suffix[11-1] – suffix[11-5-1] = suffix[10] – suffix[5] = 98 * 1015 + 97 * 1016 + 97 * 1017 + 97 * 1018 + 98 * 1019 = 108242031437886501387

Now there doesn’t seem to be any relation between these two weird integers – 1040184646587 and 108242031437886501387

Think again. Is there any relation between these two massive integers ?

Yes, there is and this observation is the core of this program/article.

1040184646587 * 101^4 = 108242031437886501387

Try thinking about this and you will find that any substring starting at index- L and ending at index- R (both inclusive) will be a palindrome if

```
(prefix[R + 1] - prefix[L]) / (101L)  =
          (suffix [n - L] - suffix [n - R- 1] ) / (101n - R - 1)
```

The rest part is just implementation.

The function computerPowers() in the program computes the powers of 101 using dynamic programming.

Overflow Issues:
As, we can see that the hash values and the reverse hash values can become huge for even the small strings of length – 8. Since C and C++ doesn’t provide support for such large numbers, so it will cause overflows. To avoid this we will take modulo of a prime (a prime number is chosen for some specific mathematical reasons). We choose the biggest possible prime which fits in an integer value. The best such value is 1000000007. Hence all the operations are done modulo 1000000007.

However Java and Python has no such issues and can be implemented without the modulo operator.

The fundamental modulo operations which are used extensively in the program are listed below.
1) Addition-
(a + b) %M = (a %M + b % M) % M
(a + b + c) % M = (a % M + b % M + c % M) % M
(a + b + c + d) % M = (a % M + b % M + c % M+ d% M) % M
…. …… …… ……
…. …… …… ……

2) Multiplication-
(a * b) % M = (a * b) % M
(a * b * c) % M = ((a * b) % M * c % M) % M
(a * b * c * d) % M = ((((a * b) % M * c) % M) * d) % M

…. …… …… ……
…. …… …… ……

This property is used by modPow() function which computes power of a number modulo M
3) Mixture of addition and multiplication-
(a * x + b * y + c) % M = ((a * x) % M +(b * y) % M+ c % M) % M

4) Subtraction-
(a – b) % M = (a % M – b % M + M) % M [Correct]
(a – b) % M = (a % M – b % M) % M [Wrong]

5) Division-
(a / b) % M = (a * MMI(b)) % M

Where MMI() is a function to calculate Modulo Multiplicative Inverse. In our program this is implemented by the function- findMMI().

```
 /* A C++ program to answer queries to check whether
   the substrings are palindrome or not efficiently */
#include<bits/stdc++.h>
using namespace std;

#define p 101
#define MOD 1000000007

// Structure to represent a query. A query consists
// of (L,R) and we have to answer whether the substring
// from index-L to R is a palindrome or not
struct Query
{
    int L, R;
};

// A function to check if a string str is palindrome
// in the ranfe L to R
bool isPalindrome(string str, int L, int R)
{
    // Keep comparing characters while they are same
    while (R > L)
        if (str[L++] != str[R--])
            return(false);
    return(true);
}

// A Function to find pow (base, exponent) % MOD
// in log (exponent) time
unsigned long long int modPow(unsigned long long int base,
                              unsigned long long int exponent)
{
    if (exponent == 0)
```

```
        return 1;
    if (exponent == 1)
        return base;

    unsigned long long int temp = modPow(base, exponent/2);

    if (exponent %2 ==0)
        return (temp%MOD * temp%MOD) % MOD;
    else
        return ((( temp%MOD * temp%MOD)%MOD) * base%MOD) % MOD;
}

// A Function to calculate Modulo Multiplicative Inverse of 'n'
unsigned long long int findMMI(unsigned long long int n)
{
    return modPow(n, MOD-2);
}

// A Function to calculate the prefix hash
void computePrefixHash(string str, int n, unsigned long long
                       int prefix[], unsigned long long int power[])
{
    prefix[0] = 0;
    prefix[1] = str[0];

    for (int i=2; i<=n; i++)
        prefix[i] = (prefix[i-1]%MOD +
                    (str[i-1]%MOD * power[i-1]%MOD)%MOD)%MOD;

    return;
}


// A Function to calculate the suffix hash
// Suffix hash is nothing but the prefix hash of
// the reversed string
void computeSuffixHash(string str, int n,
                       unsigned long long int suffix[],
                       unsigned long long int power[])
{
    suffix[0] = 0;
    suffix[1] = str[n-1];

    for (int i=n-2, j=2; i>=0 && j<=n; i--,j++)
        suffix[j] = (suffix[j-1]%MOD +
                     (str[i]%MOD * power[j-1]%MOD)%MOD)%MOD;
    return;
}
```

```
// A Function to answer the Queries
void queryResults(string str, Query q[], int m, int n,
                  unsigned long long int prefix[],
                  unsigned long long int suffix[],
                  unsigned long long int power[])
{
    for (int i=0; i<=m-1; i++)
    {
        int L = q[i].L;
        int R = q[i].R;

        // Hash Value of Substring [L,R]
        unsigned long long hash_LR =
            ((prefix[R+1]-prefix[L]+MOD)%MOD *
             findMMI(power[L])%MOD)%MOD;

        // Reverse Hash Value of Substring [L,R]
        unsigned long long reverse_hash_LR =
            ((suffix[n-L]-suffix[n-R-1]+MOD)%MOD *
             findMMI(power[n-R-1])%MOD)%MOD;

        // If both are equal then the substring is a palindrome
        if (hash_LR == reverse_hash_LR )
        {
            if (isPalindrome(str, L, R) == true)
                printf("The Substring [%d %d] is a "
                       "palindrome\n", L, R);
            else
                printf("The Substring [%d %d] is not a "
                       "palindrome\n", L, R);
        }

        else
            printf("The Substring [%d %d] is not a "
                   "palindrome\n", L, R);
    }

    return;
}

// A Dynamic Programming Based Approach to compute the
// powers of 101
void computePowers(unsigned long long int power[], int n)
{
    // 101^0 = 1
    power[0] = 1;
```

```
    for(int i=1; i<=n; i++)
        power[i] = (power[i-1]%MOD * p%MOD)%MOD;

    return;
}

/* Driver program to test above function */
int main()
{
    string str = "abaaabaaaba";
    int n = str.length();

    // A Table to store the powers of 101
    unsigned long long int power[n+1];

    computePowers(power, n);

    // Arrays to hold prefix and suffix hash values
    unsigned long long int prefix[n+1], suffix[n+1];

    // Compute Prefix Hash and Suffix Hash Arrays
    computePrefixHash(str, n, prefix, power);
    computeSuffixHash(str, n, suffix, power);

    Query q[] = {{0, 10}, {5, 8}, {2, 5}, {5, 9}};
    int m = sizeof(q)/sizeof(q[0]);

    queryResults(str, q, m, n, prefix, suffix, power);
    return (0);
}
```

Output :

```
The Substring [0 10] is a palindrome
The Substring [5 8] is not a palindrome
The Substring [2 5] is not a palindrome
The Substring [5 9] is a palindrome
```

Source

https://www.geeksforgeeks.org/palindrome-substring-queries/

Chapter 239

Parsing Apache access log in Java

Parsing Apache access log in Java - GeeksforGeeks

Web server log which maintains a history of page requests, typically appended to the end of the file. Information about the request, including client IP address, request date/time, page requested, HTTP code, bytes served, user agent, and referrer are typically added.

Given a web server log records, find the total number of successful HTTL responses (200 code) for IP addresses with successful responses.

Examples:

```
Input : Sample Access Log
192.168.1.2 - - [17/Sep/2013:22:18:19 -0700] "GET /abc HTTP/1.1" 404 201
192.168.1.2 - - [17/Sep/2013:22:18:19 -0700] "GET /favicon.ico HTTP/1.1" 200 1406
192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/ HTTP/1.1" 200 5325
192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/style.css?ve
192.168.1.3 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/js/navigatio

Output :
192.168.1.3 1
192.168.1.2 3
```

Prerequisite : Regular Expression in Java

```
 // Java program to count the no. of IP address
// count for successful http response 200 code.
import java.io.*;
import java.util.*;
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;

class FindSuccessIpCount {

    public static void findSuccessIpCount(String record)
    {
        // Creating a regular expression for the records
        final String regex = "^(\\S+) (\\S+) (\\S+) " +
               "\\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(\\S+)" +
               " (\\S+)\\s*(\\S+)?\\s*\" (\\d{3}) (\\S+)";

        final Pattern pattern = Pattern.compile(regex, Pattern.MULTILINE);
        final Matcher matcher = pattern.matcher(record);

        // Creating a Hashmap containing string as
        // the key and integer as the value.
        HashMap<String, Integer> countIP = new HashMap<String, Integer>();
        while (matcher.find()) {

            String IP = matcher.group(1);
            String Response = matcher.group(8);
            int response = Integer.parseInt(Response);

            // Inserting the IP addresses in the
            // HashMap and maintaining the frequency
            // for each HTTP 200 code.
            if (response == 200) {
                if (countIP.containsKey(IP)) {
                    countIP.put(IP, countIP.get(IP) + 1);
                }
                else {
                    countIP.put(IP, 1);
                }
            }
        }

        // Printing the hashmap
        for (Map.Entry entry : countIP.entrySet()) {
            System.out.println(entry.getKey() + " " + entry.getValue());
        }
    }
    public static void main(String[] args)
    {
        final String log = "123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] \"GET /pics/wpaper.g
                         + "123.123.123.123 - - [26/Apr/2000:00:23:47 -0400] \"GET /asctortf/ H
                         + "123.123.123.124 - - [26/Apr/2000:00:23:48 -0400] \"GET /pics/5star2
                         + "123.123.123.123 - - [26/Apr/2000:00:23:50 -0400] \"GET /pics/5star
                         + "123.123.123.126 - - [26/Apr/2000:00:23:51 -0400] \"GET /pics/a2hlog
```

```
                        + "123.123.123.123 - - [26/Apr/2000:00:23:51 -0400] \"GET /cgi-bin/ne

        findSuccessIpCount(log);
    }
}
```

Output:

```
123.123.123.126 1
123.123.123.124 1
123.123.123.123 3
```

Source

https://www.geeksforgeeks.org/parsing-apache-access-log-in-java/

Chapter 240

Postorder traversal of Binary Tree without recursion and without stack

Postorder traversal of Binary Tree without recursion and without stack - GeeksforGeeks

Prerequisite – Inorder/preorder/postorder traversal of tree
Given a binary tree, perform postorder traversal.

We have discussed below methods for postorder traversal.
1) Recursive Postorder Traversal.
2) Postorder traversal using Stack.
2) Postorder traversal using two Stacks.

In this method a DFS based solution is discussed. We keep track of visited nodes in a hash table.

```
 // CPP program or postorder traversal
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node {
    int data;
    struct Node *left, *right;
};

/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
void postorder(struct Node* head)
{
```

```
    struct Node* temp = head;
    unordered_set<Node*> visited;
    while (temp && visited.find(temp) == visited.end()) {

        // Visited left subtree
        if (temp->left &&
            visited.find(temp->left) == visited.end())
            temp = temp->left;

        // Visited right subtree
        else if (temp->right &&
                 visited.find(temp->right) == visited.end())
            temp = temp->right;

        // Print node
        else {
            printf("%d ", temp->data);
            visited.insert(temp);
            temp = head;
        }
    }
}

struct Node* newNode(int data)
{
    struct Node* node = new Node;
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return (node);
}

/* Driver program to test above functions*/
int main()
{
    struct Node* root = newNode(8);
    root->left = newNode(3);
    root->right = newNode(10);
    root->left->left = newNode(1);
    root->left->right = newNode(6);
    root->left->right->left = newNode(4);
    root->left->right->right = newNode(7);
    root->right->right = newNode(14);
    root->right->right->left = newNode(13);
    postorder(root);
    return 0;
}
```

Output:

```
1 4 7 6 3 13 14 10 8
```

Alternate Solution:
We can keep visited flag with every node instead of separate hash table.

```
 // CPP program or postorder traversal
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node {
    int data;
    struct Node *left, *right;
    bool visited;
};

void postorder(struct Node* head)
{
    struct Node* temp = head;
    while (temp && temp->visited == false) {

        // Visited left subtree
        if (temp->left && temp->left->visited == false)
            temp = temp->left;

        // Visited right subtree
        else if (temp->right && temp->right->visited == false)
            temp = temp->right;

        // Print node
        else {
            printf("%d ", temp->data);
            temp->visited = true;
            temp = head;
        }
    }
}

struct Node* newNode(int data)
{
    struct Node* node = new Node;
    node->data = data;
    node->left = NULL;
    node->right = NULL;
```

```
    node->visited = false;
    return (node);
}

/* Driver program to test above functions*/
int main()
{
    struct Node* root = newNode(8);
    root->left = newNode(3);
    root->right = newNode(10);
    root->left->left = newNode(1);
    root->left->right = newNode(6);
    root->left->right->left = newNode(4);
    root->left->right->right = newNode(7);
    root->right->right = newNode(14);
    root->right->right->left = newNode(13);
    postorder(root);
    return 0;
}
```

Output:

```
1 4 7 6 3 13 14 10 8
```

Time complexity of above solution is $O(n^2)$ in worst case we move pointer back to head after visiting every node.

Source

https://www.geeksforgeeks.org/postorder-traversal-binary-tree-without-recursion-without-stack/

Chapter 241

Practice Problems on Hashing

Practice Problems on Hashing - GeeksforGeeks

In this article, we will discuss the types of questions based on hashing. Before understanding this, you should have idea about hashing, hash function, open addressing and chaining techniques (see: Introduction, Separate chaining, Open addressing).

These are some key points in hashing:

- The purpose of hashing is to achieve search, insert and delete complexity to O(1).
- Hash function is designed to distribute keys uniformly over the hash table.
- Load factor in hash table can be defined as number of slots in hash table to number of keys to be inserted.
- For open addressing, load factor is always less than one.
- The complexity of insertion, deletion and searching using open addressing is 1/(1-).
- The complexity of insertion, deletion and searching using chaining method is (1+).

These are the types of questions asked in hashing.

Type 1: Calculation of hash values for given keys –
In this type of questions, hash values are computed by applying given hash function on given keys.

Que – 1. Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function x mod 10, which of the following statements are true? (GATE CS 2004)
i. 9679, 1989, 4199 hash to the same value
ii. 1471, 6171 has to the same value
iii. All elements hash to the same value
iv. Each element hashes to a different value
(A) i only
(B) ii only
(C) i and ii only
(D) iii or iv

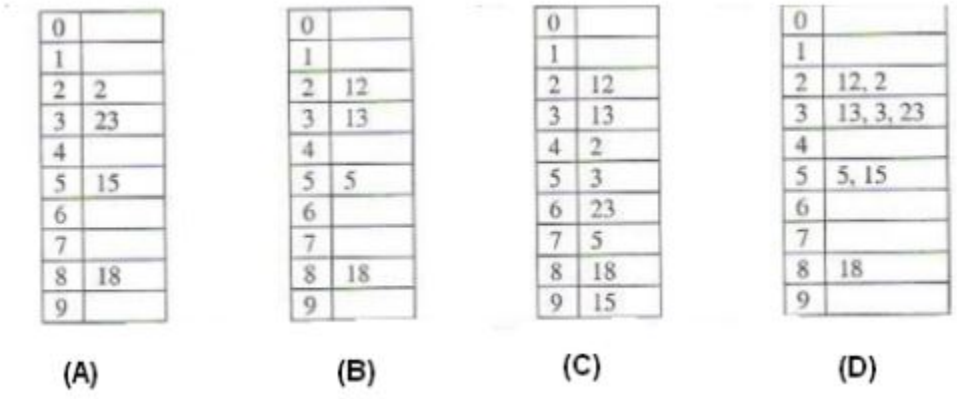
Solutions: Using given hash function h(x) = x mod 10

```
h(9679) = 9679%10 = 9
h(1989) = 1989%10 = 9
h(4199) = 4199%10 = 9
h(1471) = 1471%10 = 1
h(6171) = 6171%10 = 1
```

As we can see, 9679, 1989 and 4199 hash to same value 9. Also, 1471 and 6171 hash to same value 1. Therefore, statement (i) and (ii) are correct which match with option (C).

Type 2: Insertion of keys into hash table using linear probing as collision resolution technique –
In linear probing technique, collision is resolved by searching linearly in the hash table until an empty location is found.

Que – 2. The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table?



Solution: Keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted in hash table as:

For key 12, h(12) is 12%10 = 2. Therefore, 12 is placed at 2nd index in the hash table.
For key 18, h(18) is 18%10 = 8. Therefore, 18 is placed at 8th index in the hash table.
For key 13, h(13) is 13%10 = 3. Therefore, 13 is placed at 3rd index in the hash table.
For key 2, h(2) is 2%10 = 2. However, index 2 is already occupied with 12. Therefore, using linear probing, 2 will be placed at index 4 as index 2 and 3 are already occupied.
For key 3, h(3) is 3%10 = 3. However, index 3 is already occupied with 13. Therefore, using linear probing, 3 will be placed at index 5 as index 3 and 4 are already occupied.
Similarly, 23, 5 and 15 will be placed at index 6, 7, 9 respectively.

Therefore, correct option is (C).

Alternative Approach: We can also solve this using elimination approach as:

Option (A) and (B) are incorrect as all keys are not inserted in hash table.
Option (D) is incorrect as some indexes in hash table have more than one key which never happens using linear probing.
Remaining option is (C) which is the answer.

Type 3: Given a hash table with keys, verify/find possible sequence of keys leading to hash table –
For a given hash table, we can verify which sequence of keys can lead to that hash table. However, to find possible sequences leading to a given hash table, we need to consider all possibilities.

Que – 3. A hash table of length 10 uses open addressing with hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Which one of the following choices gives a possible order in which the key values could have been inserted in the table?
(A) 46, 42, 34, 52, 23, 33
(B) 34, 42, 23, 52, 33, 46
(C) 46, 34, 42, 23, 52, 33
(D) 42, 46, 33, 23, 34, 52

Solution: We will check whether sequence given in option A can lead to hash table given in question. Option A inserts 46, 42, 34, 52, 23, 33 as:

For key 46, h(46) is 46%10 = 6. Therefore, 46 is placed at 6th index in the hash table.
For key 42, h(42) is 42%10 = 2. Therefore, 42 is placed at 2nd index in the hash table.
For key 34, h(34) is 34%10 = 4. Therefore, 34 is placed at 4th index in the hash table.
For key 52, h(52) is 52%10 = 2. However, index 2 is occupied with 42. Therefore, 52 is placed at 3rd index in the hash table. But in given hash table, 52 is placed at 5th index. Therefore, sequence in option A can't generate hash table given in question.

In the similar way, we can check for other options as well which leads to answer as (C).

Que – 4. How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table given in Question 3 above?
(A) 10
(B) 20
(C) 30
(D) 40

Solution: The first key which is not at the index computed by hash function is 52. It means index 2, 3 and 4 were already occupied and therefore, key 52 is placed at index 5.

The keys 42, 23 and 34 are present at index 2, 3, and 4 respectively. As these keys are at their correct position, their order of insertion does not matter. These 3 keys can be inserted in 3! = 6 ways. Therefore, the sequence will be any order of (42, 23, 34) followed by 52.

The next key which is not at the index computed by hash function is 33. It means indexes 3 to 6 were already occupied and key 33 is placed at index 7. Therefore, it is the last key to be inserted into hash table.

The key 46 is present at its correct position computed by hash function. Therefore, it can be inserted at any place in the sequence before 33. The sequence excluding 33 has 4 elements 42, 23, 34, 52 which create 5 positions for 46 (3 in-between and 2 corners).
Total number of ways is: 6*5 =30

Type 4: Chaining based collision resolution technique –
In chaining based collision resolution technique, the keys generating same hash value are placed in same bucket using pointers. The different types of questions based on chaining technique are:

Que – 5. Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions? (GATE-CS-2014)
(A) (97 × 97 × 97)/100^3
(B) (99 × 98 × 97)/100^3
(C) (97 × 96 × 95)/100^3
(D) (97 × 96 × 95)/(3! × 100^3)

Solution: In uniform hashing, the function evenly distributes keys into slots of hash table. Also, each key has an equal probability of being placed into a slot, being independent of the other elements already placed.

Therefore, the probability of remaining first 3 slots empty for first insertion (choosing 4 to 100 slot) = 97/100. As next insertions are independent on previous insertion, the probability for next insertions will also be 97/100. The required probability will be (97/100)^3.

Que – 6. Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020? (GATE-CS-2015)
(A) h(i) =i^2 mod 10
(B) h(i) =i^3 mod 10
(C) h(i) = (11 ∗ i^2) mod 10
(D) h(i) = (12 ∗ i) mod 10

Solution: In uniform distribution, the function evenly distributes keys into slots of hash table.
For given hash functions, we have calculated hash values for keys 0 to 9 as:

Key	i^2 mod 10	i^3 mod 10	$(11*i^2)$mod 10	(12*i)mod 10
0	0	0	0	0
1	1	1	1	2
2	4	8	4	4
3	9	7	9	6
4	6	4	6	8
5	5	5	5	0
6	6	6	6	2
7	9	3	9	4
8	4	2	4	6
9	1	9	1	8

As we can see from the table, i^3 mod10 is distributing evenly from indexes 0 to 9. Other functions have not utilized all indexes.

Source

https://www.geeksforgeeks.org/practice-problems-on-hashing/

Chapter 242

Preorder from Inorder and Postorder traversals

Preorder from Inorder and Postorder traversals - GeeksforGeeks

Given Inorder and Postorder traversals of a binary tree, print Preorder traversal.

Example:

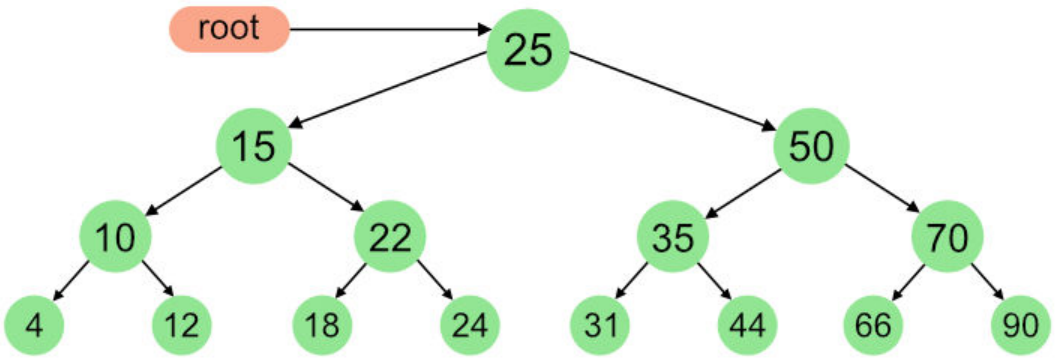
```
Input: Postorder traversal post[] = {4, 5, 2, 6, 3, 1}
       Inorder traversal in[] = {4, 2, 5, 1, 3, 6}
Output: Preorder traversal 1, 2, 4, 5, 3, 6

Trversals in the above example represents following tree
         1
      /    \
     2       3
   /   \      \
  4     5      6
```

A **naive method** is to first construct the tree from given postorder and inorder, then use simple recursive method to print preorder traversal of the constructed tree.

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



We can print preorder traversal without constructing the tree. The idea is, root is always the first item in preorder traversal and it must be the last item in postorder traversal. We first push right subtree to a stack, then left subtree and finally we push root. Finally we print contents of stack. To find boundaries of left and right subtrees in post[] and in[], we search root in in[], all elements before root in in[] are elements of left subtree and all elements after root are elements of right subtree. In post[], all elements after index of root in in[] are elements of right subtree. And elements before index (including the element at index and excluding the first element) are elements of left subtree.

```
 // Java program to print Postorder traversal from given
// Inorder and Preorder traversals.
import java.util.Stack;

public class PrintPre {

    static int postIndex;

    // Fills preorder traversal of tree with given
    // inorder and postorder traversals in a stack
    void fillPre(int[] in, int[] post, int inStrt,
```

```
            int inEnd, Stack<Integer> s)
{
    if (inStrt > inEnd)
        return;

    // Find index of next item in postorder traversal in
    // inorder.
    int val = post[postIndex];
    int inIndex = search(in, val);
    postIndex--;

    // traverse right tree
    fillPre(in, post, inIndex + 1, inEnd, s);

    // traverse left tree
    fillPre(in, post, inStrt, inIndex - 1, s);

    s.push(val);
}

// This function basically initializes postIndex
// as last element index, then fills stack with
// reverse preorder traversal using printPre
void printPreMain(int[] in, int[] post)
{
    int len = in.length;
    postIndex = len - 1;
    Stack<Integer> s = new Stack<Integer>();
    fillPre(in, post, 0, len - 1, s);
    while (s.empty() == false)
        System.out.print(s.pop() + " ");
}

// A utility function to search data in in[]
int search(int[] in, int data)
{
    int i = 0;
    for (i = 0; i < in.length; i++)
        if (in[i] == data)
            return i;
    return i;
}

// Driver code
public static void main(String ars[])
{
    int in[] = { 4, 10, 12, 15, 18, 22, 24, 25,
                 31, 35, 44, 50, 66, 70, 90 };
```

```
        int post[] = { 4, 12, 10, 18, 24, 22, 15, 31,
                       44, 35, 66, 90, 70, 50, 25 };
        PrintPre tree = new PrintPre();
        tree.printPreMain(in, post);
    }
}
```

Output:

```
25 15 10 4 12 22 18 24 50 35 31 44 70 66 90
```

Time Complexity: The above function visits every node in array. For every visit, it calls search which takes O(n) time. Therefore, overall time complexity of the function is $O(n^2)$

O(n) Solution

We can further optimize above solution to first hash all items of inorder traversal so that we do not have to linearly search items. With hash table available to us, we can search an item in O(1) time.

```
 // Java program to print Postorder traversal from given
// Inorder and Preorder traversals.
import java.util.Stack;
import java.util.HashMap;

public class PrintPre {

    static int postIndex;

    // Fills preorder traversal of tree with given
    // inorder and postorder traversals in a stack
    void fillPre(int[] in, int[] post, int inStrt, int inEnd,
                 Stack<Integer> s, HashMap<Integer, Integer> hm)
    {
        if (inStrt > inEnd)
            return;

        // Find index of next item in postorder traversal in
        // inorder.
        int val = post[postIndex];
        int inIndex = hm.get(val);
        postIndex--;

        // traverse right tree
        fillPre(in, post, inIndex + 1, inEnd, s, hm);

        // traverse left tree
```

```
        fillPre(in, post, inStrt, inIndex - 1, s, hm);

        s.push(val);
    }

    // This function basically initializes postIndex
    // as last element index, then fills stack with
    // reverse preorder traversal using printPre
    void printPreMain(int[] in, int[] post)
    {
        int len = in.length;
        postIndex = len - 1;
        Stack<Integer> s = new Stack<Integer>();

        // Insert values in a hash map and their indexes.
        HashMap<Integer, Integer> hm =
                    new HashMap<Integer, Integer>();
        for (int i = 0; i < in.length; i++)
            hm.put(in[i], i);

        // Fill preorder traversal in a stack
        fillPre(in, post, 0, len - 1, s, hm);

        // Print contents of stack
        while (s.empty() == false)
            System.out.print(s.pop() + " ");
    }

    // Driver code
    public static void main(String ars[])
    {
        int in[] = { 4, 10, 12, 15, 18, 22, 24, 25,
                    31, 35, 44, 50, 66, 70, 90 };
        int post[] = { 4, 12, 10, 18, 24, 22, 15, 31,
                       44, 35, 66, 90, 70, 50, 25 };
        PrintPre tree = new PrintPre();
        tree.printPreMain(in, post);
    }
}
```

Output:

```
25 15 10 4 12 22 18 24 50 35 31 44 70 66 90
```

Time Complexity: O(n)

Source

https://www.geeksforgeeks.org/preorder-from-inorder-and-postorder-traversals/

Chapter 243

Print All Distinct Elements of a given integer array

Print All Distinct Elements of a given integer array - GeeksforGeeks

Given an integer array, print all distinct elements in array. The given array may contain duplicates and the output should print every element only once. The given array is not sorted.

Examples:

```
Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45}
Output: 12, 10, 9, 45, 2

Input: arr[] = {1, 2, 3, 4, 5}
Output: 1, 2, 3, 4, 5

Input: arr[] = {1, 1, 1, 1, 1}
Output: 1
```

A **Simple Solution** is to use twp nested loops. The outer loop picks an element one by one starting from the leftmost element. The inner loop checks if the element is present on left side of it. If present, then ignores the element, else prints the element. Following is the implementation of the simple algorithm.

C++

```
 // C++ program to print all distinct elements in a given array
#include <iostream>
#include <algorithm>
using namespace std;
```

```
void printDistinct(int arr[], int n)
{
    // Pick all elements one by one
    for (int i=0; i<n; i++)
    {
        // Check if the picked element is already printed
        int j;
        for (j=0; j<i; j++)
           if (arr[i] == arr[j])
               break;

        // If not printed earlier, then print it
        if (i == j)
          cout << arr[i] << " ";
    }
}

// Driver program to test above function
int main()
{
    int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    printDistinct(arr, n);
    return 0;
}
```

Java

```
 // Java program to print all distinct
// elements in a given array
import java.io.*;

class GFG {

    static void printDistinct(int arr[], int n)
    {
        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {
            // Check if the picked element
            // is already printed
            int j;
            for (j = 0; j < i; j++)
            if (arr[i] == arr[j])
                break;

            // If not printed earlier,
```

```
            // then print it
            if (i == j)
            System.out.print( arr[i] + " ");
        }
    }

    // Driver program
    public static void main (String[] args)
    {
        int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
        int n = arr.length;
        printDistinct(arr, n);

    }
}

// This code is contributed by vt_m
```

Python3

```
 # python program to print all distinct
# elements in a given array

def printDistinct(arr, n):

    # Pick all elements one by one
    for i in range(0, n):

        # Check if the picked element
        # is already printed
        d = 0
        for j in range(0, i):
            if (arr[i] == arr[j]):
                d = 1
                break

        # If not printed earlier,
        # then print it
        if (d == 0):
            print(arr[i])

# Driver program to test above function
arr = [6, 10, 5, 4, 9, 120, 4, 6, 10]
n = len(arr)
printDistinct(arr, n)

# This code is contributed by Sam007.
```

C#

```
 // C# program to print all distinct
// elements in a given array
using System;

class GFG {

    static void printDistinct(int []arr, int n)
    {

        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {

            // Check if the picked element
            // is already printed
            int j;
            for (j = 0; j < i; j++)
                if (arr[i] == arr[j])
                     break;

            // If not printed earlier,
            // then print it
            if (i == j)
            Console.Write(arr[i] + " ");
        }
    }

    // Driver program
    public static void Main ()
    {
        int []arr = {6, 10, 5, 4, 9, 120,
                                  4, 6, 10};
        int n = arr.Length;

        printDistinct(arr, n);

    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to print all distinct
```

```
// elements in a given array

function printDistinct($arr, $n)
{
    // Pick all elements one by one
    for($i = 0; $i < $n; $i++)
    {

        // Check if the picked element
        // is already printed
        $j;
        for($j = 0; $j < $i; $j++)
        if ($arr[$i] == $arr[$j])
            break;

        // If not printed
        // earlier, then print it
        if ($i == $j)
        echo $arr[$i] , " ";
    }
}

    // Driver Code
    $arr = array(6, 10, 5, 4, 9, 120, 4, 6, 10);
    $n = sizeof($arr);
    printDistinct($arr, $n);

// This code is contributed by nitin mittal
?>
```

Output:

```
6 10 5 4 9 120
```

Time Complexity of above solution is O(n^2). We can **Use Sorting** to solve the problem in O(nLogn) time. The idea is simple, first sort the array so that all occurrences of every element become consecutive. Once the occurrences become consecutive, we can traverse the sorted array and print distinct elements in O(n) time. Following is the implementation of the idea.

C++

```
 // C++ program to print all distinct elements in a given array
#include <iostream>
#include <algorithm>
using namespace std;
```

```
void printDistinct(int arr[], int n)
{
    // First sort the array so that all occurrences become consecutive
    sort(arr, arr + n);

    // Traverse the sorted array
    for (int i=0; i<n; i++)
    {
       // Move the index ahead while there are duplicates
       while (i < n-1 && arr[i] == arr[i+1])
          i++;

       // print last occurrence of the current element
       cout << arr[i] << " ";
    }
}

// Driver program to test above function
int main()
{
    int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    printDistinct(arr, n);
    return 0;
}
```

Java

```
 // Java program to print all distinct
// elements in a given array
import java.io.*;
import java .util.*;

class GFG
{
    static void printDistinct(int arr[], int n)
    {
        // First sort the array so that
        // all occurrences become consecutive
        Arrays.sort(arr);

        // Traverse the sorted array
        for (int i = 0; i < n; i++)
        {
            // Move the index ahead while
            // there are duplicates
            while (i < n - 1 && arr[i] == arr[i + 1])
                i++;
```

```
            // print last occurrence of
            // the current element
            System.out.print(arr[i] +" ");
        }
    }

    // Driver program
    public static void main (String[] args)
    {
        int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
        int n = arr.length;
        printDistinct(arr, n);

    }
}

// This code is contributed by vt_m
```

C#

```
 // C# program to print all distinct
// elements in a given array
using System;

class GFG {

    static void printDistinct(int []arr, int n)
    {

        // First sort the array so that
        // all occurrences become consecutive
        Array.Sort(arr);

        // Traverse the sorted array
        for (int i = 0; i < n; i++)
        {

            // Move the index ahead while
            // there are duplicates
            while (i < n - 1 && arr[i] == arr[i + 1])
                i++;

            // print last occurrence of
            // the current element
            Console.Write(arr[i] + " ");
        }
    }
```

```
    // Driver program
    public static void Main ()
    {
        int []arr = {6, 10, 5, 4, 9, 120, 4, 6, 10};
        int n = arr.Length;

        printDistinct(arr, n);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to print all distinct
// elements in a given array

function printDistinct( $arr, $n)
{

    // First sort the array so
    // that all occurrences
    // become consecutive
    sort($arr);

    // Traverse the sorted array
    for ($i = 0; $i < $n; $i++)
    {

        // Move the index ahead
        // while there are duplicates
        while ($i < $n - 1 and
               $arr[$i] == $arr[$i + 1])
            $i++;

        // print last occurrence
        // of the current element
        echo $arr[$i] , " ";
    }
}

    // Driver Code
    $arr = array(6, 10, 5, 4, 9, 120, 4, 6, 10);
    $n = count($arr);
    printDistinct($arr, $n);
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
4 5 6 9 10 120
```

We can **Use Hashing** to solve this in O(n) time on average. The idea is to traverse the given array from left to right and keep track of visited elements in a hash table. Following is Java implementation of the idea.

```
 /* Java program to print all distinct elements of a given array */
import java.util.*;

class Main
{
    // This function prints all distinct elements
    static void printDistinct(int arr[])
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array
        for (int i=0; i<arr.length; i++)
        {
            // If not present, then put it in hashtable and print it
            if (!set.contains(arr[i]))
            {
                set.add(arr[i]);
                System.out.print(arr[i] + " ");
            }
        }
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        printDistinct(arr);
    }
}
```

Output:

```
10 5 3 4 6
```

One more advantage of hashing over sorting is, the elements are printed in same order as they are in input array.

Improved By : Sam007, nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/print-distinct-elements-given-integer-array/

Chapter 244

Print Longest substring without repeating characters

Print Longest substring without repeating characters - GeeksforGeeks

Given a string, print the longest substring without repeating characters. For example, the longest substrings without repeating characters for "ABDEFGABEF" are "BDEFGA" and "DEFGAB", with length 6. For "BBBB" the longest substring is "B", with length 1. The desired time complexity is O(n) where n is the length of the string.

Prerequisite: Length of longest substring without repeating characters

```
Examples:
Input : GEEKSFORGEEKS
Output : EKSFORG

Input : ABDEFGABEF
Output : BDEFGA
```

Approach: The idea is to traverse the string and for each already visited character store its last occurrence in a hash table(Here unordered_map is used as hash with key as character and value as its last position). The variable st stores starting point of current substring, maxlen stores length of maximum length substring and start stores starting index of maximum length substring. While traversing the string, check whether current character is present in hash table or not. If it is not present, then store current character in hash table with value as current index. If it is already present in hash table, this means the current character could repeat in current substring. For this check if the previous occurrence of character is before or after the starting point st of current substring. If it is before st, then only update the value in hash table. If it is after st, then find length of current substring currlen as i-st, where i is current index. Compare currlen with maxlen. If maxlen is less than currlen, then update maxlen as currlen and start as st. After complete traversal of string, the required longest substring without repeating characters is from s[start] to s[start+maxlen-1].

Implementation:

```
 // C++ program to find and print longest
// substring without repeating characters.
#include <bits/stdc++.h>

using namespace std;

// Function to find and print longest
// substring without repeating characters.
string findLongestSubstring(string str)
{
    int i;
    int n = str.length();

    // starting point of current substring.
    int st = 0;

    // length of current substring.
    int currlen;

    // maximum length substring without repeating
    // characters.
    int maxlen = 0;

    // starting index of maximum length substring.
    int start;

    // Hash Map to store last occurrence of each
    // already visited character.
    unordered_map<char, int> pos;

    // Last occurrence of first character is index 0;
    pos[str[0]] = 0;

    for (i = 1; i < n; i++) {

        // If this character is not present in hash,
        // then this is first occurrence of this
        // character, store this in hash.
        if (pos.find(str[i]) == pos.end())
            pos[str[i]] = i;

        else {
            // If this character is present in hash then
            // this character has previous occurrence,
            // check if that occurrence is before or after
            // starting point of current substring.
```

```
            if (pos[str[i]] >= st) {

                // find length of current substring and
                // update maxlen and start accordingly.
                currlen = i - st;
                if (maxlen < currlen) {
                    maxlen = currlen;
                    start = st;
                }

                // Next substring will start after the last
                // occurrence of current character to avoid
                // its repetition.
                st = pos[str[i]] + 1;
            }

            // Update last occurrence of
            // current character.
            pos[str[i]] = i;
        }
    }

    // Compare length of last substring with maxlen and
    // update maxlen and start accordingly.
    if (maxlen < i - st) {
        maxlen = i - st;
        start = st;
    }

    // The required longest substring without
    // repeating characters is from str[start]
    // to str[start+maxlen-1].
    return str.substr(start, maxlen);
}

// Driver function
int main()
{
    string str = "GEEKSFORGEEKS";
    cout << findLongestSubstring(str);
    return 0;
}


Output: EKSFORG
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Source

https://www.geeksforgeeks.org/print-longest-substring-without-repeating-characters/

Chapter 245

Print Nodes in Top View of Binary Tree

Print Nodes in Top View of Binary Tree - GeeksforGeeks

Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. Given a binary tree, print the top view of it. The output nodes can be printed in any order. Expected time complexity is O(n)

A node x is there in output if x is the topmost node at its horizontal distance. Horizontal distance of left child of a node x is equal to horizontal distance of x minus 1, and that of right child is horizontal distance of x plus 1.

```
       1
    /     \
   2       3
  /  \    / \
 4    5  6   7
Top view of the above binary tree is
4 2 1 3 7
```

```
        1
      /   \
    2       3
      \
        4
          \
            5
             \
               6
Top view of the above binary tree is
2 1 3 6
```

The idea is to do something similar to vertical Order Traversal. Like vertical Order Traversal, we need to nodes of same horizontal distance together. We do a level order traversal so that the topmost node at a horizontal node is visited before any other node of same horizontal distance below it. Hashing is used to check if a node at given horizontal distance is seen or not.

C++

```
 // C++ program to print top
// view of binary tree
#include <bits/stdc++.h>
using namespace std;

// Structure of binary tree
struct Node {
    int data;
    struct Node *left, *right;
};

// function should print the topView of
// the binary tree
void topView(struct Node* root)
{
    if (root == NULL)
        return;

    unordered_map<int, int> m;
    queue<pair<Node*, int> > q;

    // push node and horizontal distance to queue
    q.push(make_pair(root, 0));

    while (!q.empty()) {
        pair<Node*, int> p = q.front();
        Node* n = p.first;
        int val = p.second;
        q.pop();

        // if horizontal value is not in the hashmap
        // that means it is the first value with that
        // horizontal distance so print it and store
        // this value in hashmap
        if (m.find(val) == m.end()) {
            m[val] = n->data;
            printf("%d ", n->data);
        }

        if (n->left != NULL)
            q.push(make_pair(n->left, val - 1));
```

```
        if (n->right != NULL)
            q.push(make_pair(n->right, val + 1));
    }
}

// function to create a new node
struct Node* newNode(int key)
{
    struct Node* node = new Node;
    node->data = key;
    node->left = node->right = NULL;
    return node;
}

// main function
int main()
{
    /* Create following Binary Tree
              1
            /  \
           2    3
            \
             4
              \
               5
                \
                 6*/

    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->right = newNode(4);
    root->left->right->right = newNode(5);
    root->left->right->right->right = newNode(6);

    topView(root);
    return 0;
}

/* This code is contributed by Niteesh Kumar */
```

Java

```
 // Java program to print top view of Binary tree
import java.util.*;

// Class for a tree node
```

```
class TreeNode {
    // Members
    int key;
    TreeNode left, right;

    // Constructor
    public TreeNode(int key)
    {
        this.key = key;
        left = right = null;
    }
}

// A class to represent a queue item. The queue is used to do Level
// order traversal. Every Queue item contains node and horizontal
// distance of node from root
class QItem {
    TreeNode node;
    int hd;
    public QItem(TreeNode n, int h)
    {
        node = n;
        hd = h;
    }
}

// Class for a Binary Tree
class Tree {
    TreeNode root;

    // Constructors
    public Tree() { root = null; }
    public Tree(TreeNode n) { root = n; }

    // This method prints nodes in top view of binary tree
    public void printTopView()
    {
        // base case
        if (root == null) {
            return;
        }

        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Create a queue and add root to it
        Queue<QItem> Q = new LinkedList<QItem>();
        Q.add(new QItem(root, 0)); // Horizontal distance of root is 0
```

```
        // Standard BFS or level order traversal loop
        while (!Q.isEmpty()) {
            // Remove the front item and get its details
            QItem qi = Q.remove();
            int hd = qi.hd;
            TreeNode n = qi.node;

            // If this is the first node at its horizontal distance,
            // then this node is in top view
            if (!set.contains(hd)) {
                set.add(hd);
                System.out.print(n.key + " ");
            }

            // Enqueue left and right children of current node
            if (n.left != null)
                Q.add(new QItem(n.left, hd - 1));
            if (n.right != null)
                Q.add(new QItem(n.right, hd + 1));
        }
    }
}

// Driver class to test above methods
public class Main {
    public static void main(String[] args)
    {
        /* Create following Binary Tree
             1
           /  \
          2    3
           \
            4
             \
              5
               \
                6*/
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(4);
        root.left.right.right = new TreeNode(5);
        root.left.right.right.right = new TreeNode(6);
        Tree t = new Tree(root);
        System.out.println("Following are nodes in top view of Binary Tree");
        t.printTopView();
    }
```

```
}
```

Output:

```
Following are nodes in top view of Binary Tree
1 2 3 6
```

Time Complexity of the above implementation is O(n) where n is number of nodes in given binary tree. The assumption here is that add() and contains() methods of HashSet work in O(1) time.

This article is contributed by **Rohan**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Aarsee

Source

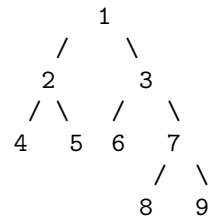
https://www.geeksforgeeks.org/print-nodes-top-view-binary-tree/

Chapter 246

Print a Binary Tree in Vertical Order | Set 2 (Map based Method)

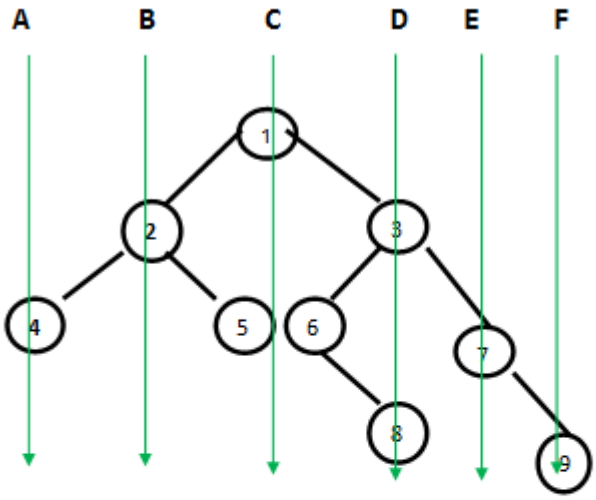
Print a Binary Tree in Vertical Order | Set 2 (Map based Method) - GeeksforGeeks

Given a binary tree, print it vertically. The following example illustrates vertical order traversal.




```
The output of print this tree vertically will be:
4
2
1 5 6
3 8
7
9
```

Vertical Lines



Vertical order traversal is:

A- 4

B- 2

C- 1 5 6

D- 3 8

E- 7

F- 9

We have discussed a $O(n^2)$ solution in the previous post. In this post, an efficient solution based on hash map is discussed. We need to check the Horizontal Distances from root for all nodes. If two nodes have the same Horizontal Distance (HD), then they are on same vertical line. The idea of HD is simple. HD for root is 0, a right edge (edge connecting to right subtree) is considered as +1 horizontal distance and a left edge is considered as -1 horizontal distance. For example, in the above tree, HD for Node 4 is at -2, HD for Node 2 is -1, HD for 5 and 6 is 0 and HD for node 7 is +2.
We can do preorder traversal of the given Binary Tree. While traversing the tree, we can recursively calculate HDs. We initially pass the horizontal distance as 0 for root. For left subtree, we pass the Horizontal Distance as Horizontal distance of root minus 1. For right subtree, we pass the Horizontal Distance as Horizontal Distance of root plus 1. For every HD value, we maintain a list of nodes in a hasp map. Whenever we see a node in traversal, we go to the hash map entry and add the node to the hash map using HD as a key in map.

Following is C++ implementation of the above method. Thanks to Chirag for providing the below C++ implementation.

C++

```
 // C++ program for printing vertical order of a given binary tree
#include <iostream>
#include <vector>
#include <map>
using namespace std;

// Structure for a binary tree node
struct Node
{
    int key;
    Node *left, *right;
};

// A utility function to create a new node
struct Node* newNode(int key)
{
    struct Node* node = new Node;
    node->key = key;
    node->left = node->right = NULL;
    return node;
}

// Utility function to store vertical order in map 'm'
// 'hd' is horigontal distance of current node from root.
// 'hd' is initally passed as 0
void getVerticalOrder(Node* root, int hd, map<int, vector<int>> &m)
{
    // Base case
    if (root == NULL)
        return;

    // Store current node in map 'm'
    m[hd].push_back(root->key);

    // Store nodes in left subtree
    getVerticalOrder(root->left, hd-1, m);

    // Store nodes in right subtree
    getVerticalOrder(root->right, hd+1, m);
}

// The main function to print vertical oder of a binary tree
// with given root
void printVerticalOrder(Node* root)
{
    // Create a map and store vertical oder in map using
```

```
    // function getVerticalOrder()
    map < int,vector<int> > m;
    int hd = 0;
    getVerticalOrder(root, hd,m);

    // Traverse the map and print nodes at every horigontal
    // distance (hd)
    map< int,vector<int> > :: iterator it;
    for (it=m.begin(); it!=m.end(); it++)
    {
        for (int i=0; i<it->second.size(); ++i)
            cout << it->second[i] << " ";
        cout << endl;
    }
}

// Driver program to test above functions
int main()
{
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);
    cout << "Vertical order traversal is n";
    printVerticalOrder(root);
    return 0;
}
```

Java

```
 // Java program for printing vertical order of a given binary tree
import java.util.TreeMap;
import java.util.Vector;
import java.util.Map.Entry;

public class VerticalOrderBtree
{
    // Tree node
    static class Node
    {
        int key;
        Node left;
        Node right;
```

```
    // Constructor
    Node(int data)
    {
        key = data;
        left = null;
        right = null;
    }
}

// Utility function to store vertical order in map 'm'
// 'hd' is horizontal distance of current node from root.
// 'hd' is initially passed as 0
static void getVerticalOrder(Node root, int hd,
                             TreeMap<Integer,Vector<Integer>> m)
{
    // Base case
    if(root == null)
        return;

    //get the vector list at 'hd'
    Vector<Integer> get =  m.get(hd);

    // Store current node in map 'm'
    if(get == null)
    {
        get = new Vector<>();
        get.add(root.key);
    }
    else
        get.add(root.key);

    m.put(hd, get);

    // Store nodes in left subtree
    getVerticalOrder(root.left, hd-1, m);

    // Store nodes in right subtree
    getVerticalOrder(root.right, hd+1, m);
}

// The main function to print vertical oder of a binary tree
// with given root
static void printVerticalOrder(Node root)
{
    // Create a map and store vertical oder in map using
    // function getVerticalOrder()
    TreeMap<Integer,Vector<Integer>> m = new TreeMap<>();
```

```
        int hd =0;
        getVerticalOrder(root,hd,m);

        // Traverse the map and print nodes at every horigontal
        // distance (hd)
        for (Entry<Integer, Vector<Integer>> entry : m.entrySet())
        {
            System.out.println(entry.getValue());
        }
    }

    // Driver program to test above functions
    public static void main(String[] args) {

        // TO DO Auto-generated method stub
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.left.right = new Node(8);
        root.right.right.right = new Node(9);
        System.out.println("Vertical Order traversal is");
        printVerticalOrder(root);
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Python program for printing vertical order of a given
# binary tree

# A binary tree node
class Node:
    # Constructor to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

# Utility function to store vertical order in map 'm'
# 'hd' is horizontal distance of current node from root
# 'hd' is initially passed as 0
def getVerticalOrder(root, hd, m):
```

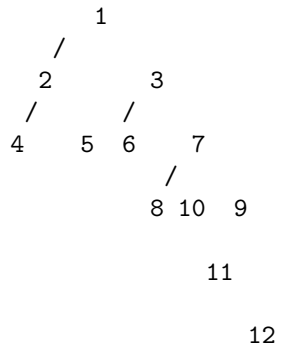
```
    # Base Case
    if root is None:
        return

    # Store current node in map 'm'
    try:
        m[hd].append(root.key)
    except:
        m[hd] = [root.key]

    # Store nodes in left subtree
    getVerticalOrder(root.left, hd-1, m)

    # Store nodes in right subtree
    getVerticalOrder(root.right, hd+1, m)

# The main function to print vertical order of a binary
#tree ith given root
def printVerticalOrder(root):

    # Create a map and store vertical order in map using
    # function getVerticalORder()
    m = dict()
    hd = 0
    getVerticalOrder(root, hd, m)

    # Traverse the map and print nodes at every horizontal
    # distance (hd)
    for index, value in enumerate(sorted(m)):
        for i in m[value]:
            print i,
        print


# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
root.right.left.right = Node(8)
root.right.right.right = Node(9)
print "Vertical order traversal is"
printVerticalOrder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Vertical order traversal is
4
2
1 5 6
3 8
7
9
```

Time Complexity of hashing based solution can be considered as O(n) under the assumption that we have good hashing function that allows insertion and retrieval operations in O(1) time. In the above C++ implementation, map of STL is used. map in STL is typically implemented using a Self-Balancing Binary Search Tree where all operations take O(Logn) time. Therefore time complexity of above implementation is O(nLogn).

Note that the above solution may print nodes in same vertical order as they appear in tree. For example, the above program prints 12 before 9. See this for a sample run.



Refer below post for level order traversal based solution. The below post makes sure that nodes of a vertical line are printed in same order as they appear in tree.

Print a Binary Tree in Vertical Order | Set 3 (Using Level Order Traversal)

Source

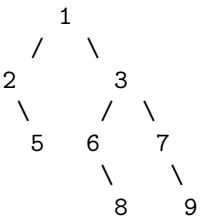
https://www.geeksforgeeks.org/print-binary-tree-vertical-order-set-2/

Chapter 247

Print a Binary Tree in Vertical Order | Set 3 (Using Level Order Traversal)

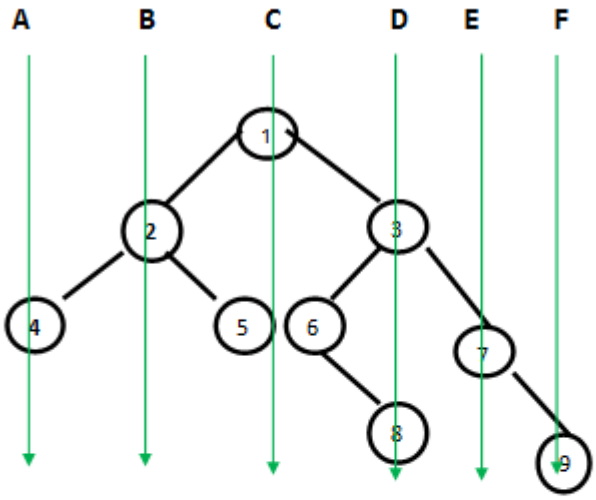
Print a Binary Tree in Vertical Order | Set 3 (Using Level Order Traversal) - GeeksforGeeks

Given a binary tree, print it vertically. The following example illustrates vertical order traversal.



```
The output of print this tree vertically will be:
4
2
1 5 6
3 8
7
9
```

Vertical Lines



Vertical order traversal is:

A- 4

B- 2

C- 1 5 6

D- 3 8

E- 7

F- 9

We have discussed an efficient approach in below post.

Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)

The above solution uses preorder traversal and Hashmap to store nodes according to horizontal distances. Since above approach uses preorder traversal, nodes in a vertical line may not be prined in same order as they appear in tree. For example, the above solution prints 12 before 9 in below tree. See this for a sample run.

```
        1
      /   \
     2     3
    /  \  /  \
   4    5 6   7
            \ /  \
            8 10  9
```

```
\
11
  \
   12
```

If we use level order traversal, we can make sure that if a node like 12 comes below in same vertical line, it is printed after a node like 9 which comes above in vertical line.

```
1. To maintain a hash for the branch of each node.
2. Traverse the tree in level order fashion.
3. In level order traversal, maintain a queue
   which holds, node and its vertical branch.
    * pop from queue.
    * add this node's data in vector corresponding
      to its branch in the hash.
    * if this node hash left child, insert in the
      queue, left with branch - 1.
    * if this node hash right child, insert in the
      queue, right with branch + 1.
```

C++

```
 // C++ program for printing vertical order
// of a given binary tree usin BFS.
#include<bits/stdc++.h>

using namespace std;

// Structure for a binary tree node
struct Node
{
    int key;
    Node *left, *right;
};

// A utility function to create a new node
Node* newNode(int key)
{
    Node* node = new Node;
    node->key = key;
    node->left = node->right = NULL;
    return node;
}

// The main function to print vertical oder of a
// binary tree with given root
```

```
void printVerticalOrder(Node* root)
{
    // Base case
    if (!root)
        return;

    // Create a map and store vertical oder in
    // map using function getVerticalOrder()
    map < int,vector<int> > m;
    int hd = 0;

    // Create queue to do level order traversal.
    // Every item of queue contains node and
    // horizontal distance.
    queue<pair<Node*, int> > que;
    que.push(make_pair(root, hd));

     while (!que.empty())
     {
        // pop from queue front
        pair<Node *,int> temp = que.front();
        que.pop();
        hd = temp.second;
        Node* node = temp.first;

        // insert this node's data in vector of hash
        m[hd].push_back(node->key);

        if (node->left != NULL)
            que.push(make_pair(node->left, hd-1));
        if (node->right != NULL)
            que.push(make_pair(node->right, hd+1));
    }

    // Traverse the map and print nodes at
    // every horigontal distance (hd)
    map< int,vector<int> > :: iterator it;
    for (it=m.begin(); it!=m.end(); it++)
    {
        for (int i=0; i<it->second.size(); ++i)
            cout << it->second[i] << " ";
        cout << endl;
    }
}

// Driver program to test above functions
int main()
{
```

```
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);
    root->right->right->left= newNode(10);
    root->right->right->left->right= newNode(11);
    root->right->right->left->right->right= newNode(12);
    cout << "Vertical order traversal is \n";
    printVerticalOrder(root);
    return 0;
}
```

Python3

```
 #python3 Program to print zigzag traversal of binary tree
import collections
# Binary tree node
class Node:
    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = self.right = None

# function to print vertical order traversal of binary tree
def verticalTraverse(root):

    # Base case
    if root is None:
        return

    # Create empty queue for level order traversal
    queue = []

    # create a map to store nodes at a particular
    # horizontal distance
    m = {}

    # map to store horizontal distance of nodes
    hd_node = {}

    # enqueue root
    queue.append(root)
    # store the horizontal distance of root as 0
```

```
    hd_node[root] = 0

    m[0] = [root.data]

    # loop will run while queue is not empty
    while len(queue) > 0:

        # dequeue node from queue
        temp = queue.pop(0)

        if temp.left:
            # Enqueue left child
            queue.append(temp.left)

            # Store the horizontal distance of left node
            # hd(left child) = hd(parent) -1
            hd_node[temp.left] = hd_node[temp] - 1
            hd = hd_node[temp.left]

            if m.get(hd) is None:
                m[hd] = []

            m[hd].append(temp.left.data)

        if temp.right:
            # Enqueue right child
            queue.append(temp.right)

            # store the horizontal distance of right child
            # hd(right child) = hd(parent) + 1
            hd_node[temp.right] = hd_node[temp] + 1
            hd = hd_node[temp.right]

            if m.get(hd) is None:
                m[hd] = []

            m[hd].append(temp.right.data)

    # Sort the map according to horizontal distance
    sorted_m = collections.OrderedDict(sorted(m.items()))

    # Traverse the sorted map and print nodes at each horizontal distance
    for i in sorted_m.values():
        for j in i:
            print(j, " ", end="")
        print()

# Driver program to check above function
```

```
"""
Constructed binary tree is
            1
        / \
        2     3
    / \ / \
    4     5 6     7
            \ / \
            8 10 9
                \
                11
                    \
                    12

"""
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
root.right.left.right =Node(8)
root.right.right.left = Node(10)
root.right.right.right = Node(9)
root.right.right.left.right = Node(11)
root.right.right.left.right.right = Node(12)
print("Vertical order traversal is ")
verticalTraverse(root)

# This code is contributed by Shweta Singh
```

Output:

```
Vertical order traversal is
4
2
1  5  6
3  8  10
7  11
9  12
```

Time Complexity of above implementation is O(n Log n). Note that above implementation uses map which is implemented using self-balancing BST.

We can reduce time complexity to O(n) using unordered_map. To print nodes in desired order, we can have 2 variables denoting min and max horizontal distance. We can simply

iterate from min to max horizontal distance and get corresponding values from Map. So it is O(n)

Auxiliary Space : O(n)

Improved By : shweta44

Source

https://www.geeksforgeeks.org/print-a-binary-tree-in-vertical-order-set-3-using-level-order-traversal/

Chapter 248

Print all Subsequences of String which Start with Vowel and End with Consonant.

Print all Subsequences of String which Start with Vowel and End with Consonant. - GeeksforGeeks

Given a string return all possible subsequences which start with vowel and end with consonant. A String is a subsequence of a given String, that is generated by deleting some character of a given string without changing its order.
Examples:

```
Input : 'abc'
Output : ab, ac, abc

Input : 'aab'
Output : ab, aab
```

Question Source: Yatra.com Interview Experience | Set 7

Explanation of the Algorithm:

```
Step 1: Iterate over the entire String
Step 2: check if the ith character for vowel
Step 3: If true iterate the string from the end,
        if false move to next iteration
Step 4: check the jth character for consonent
        if false move to next iteration
        if true perform the following
```

```
Step 5: Add the substring starting at index i and
        ending at index j to the hastset.
Step 6: Iterate over the substring drop each character
        and recur to generate all its subString


 // Java Program to generate all the subsequence
// starting with vowel and ending with consonant.
import java.util.HashSet;

public class Subsequence {

    // Set to store all the subsequences
    static HashSet<String> st = new HashSet<>();

    // It computes all the possible substring that
    // starts with vowel and end with consonent
    static void subsequence(String str)
    {
        // iterate over the entire string
        for (int i = 0; i < str.length(); i++) {

            // test ith character for vowel
            if (isVowel(str.charAt(i))) {

                // if the ith character is vowel
                // iterate from end of the string
                // and check for consonant.
                for (int j = (str.length() - 1); j >= i; j--) {

                    // test jth character for consonant.
                    if (isConsonant(str.charAt((j)))) {

                        // once we get a consonant add it to
                        // the hashset
                        String str_sub = str.substring(i, j + 1);
                        st.add(str_sub);

                        // drop each character of the substring
                        // and recur to generate all subsquence
                        // of the substring
                        for (int k = 1; k < str_sub.length() - 1; k++) {
                            StringBuffer sb = new StringBuffer(str_sub);
                            sb.deleteCharAt(k);
                            subsequence(sb.toString());
                        }
                    }
                }
            }
```

```
        }
    }

    // Utility method to check vowel
    static boolean isVowel(char c)
    {
        return (c == 'a' || c == 'e' || c == 'i' || c == 'o'
                                                 || c == 'u');
    }

    // Utility method to check consonant
    static boolean isConsonant(char c)
    {
        return !(c == 'a' || c == 'e' || c == 'i' || c == 'o'
                                                 || c == 'u');
    }

    // Driver code
    public static void main(String[] args)
    {
        String s = "xabcef";
        subsequence(s);
        System.out.println(st);
    }
}
```

Output:

```
[ef, ab, ac, aef, abc, abf, af, acf, abcef, abcf, acef, abef]
```

Source

https://www.geeksforgeeks.org/subsequences-string-start-vowel-end-consonant/

Chapter 249

Print all pairs with given sum

Print all pairs with given sum - GeeksforGeeks

Given an array of integers, and a number 'sum', print all pairs in the array whose sum is equal to 'sum'.

```
Examples :
Input  :  arr[] = {1, 5, 7, -1, 5},
          sum = 6
Output : (1, 5) (7, -1) (1, 5)

Input  :  arr[] = {2, 5, 17, -1},
          sum = 7
Output :  (2, 5)
```

A **simple solution** is be traverse each element and check if there's another number in the array which can be added to it to give sum.

C++

```
 // C++ implementation of simple method to
// find print pairs with given sum.
#include <bits/stdc++.h>
using namespace std;

// Returns number of pairs in arr[0..n-1]
// with sum equal to 'sum'
int printPairs(int arr[], int n, int sum)
{
    int count = 0; // Initialize result

    // Consider all possible pairs and check
```

```
    // their sums
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] + arr[j] == sum)
                cout << "(" << arr[i] << ", "
                     << arr[j] << ")" << endl;
}

// Driver function to test the above function
int main()
{
    int arr[] = { 1, 5, 7, -1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int sum = 6;
    printPairs(arr, n, sum);
    return 0;
}
```

Java

```
 // Java implementation of
// simple method to find
// print pairs with given sum.

class GFG
{

// Returns number of pairs
// in arr[0..n-1] with sum
// equal to 'sum'
static void printPairs(int arr[],
                       int n, int sum)
{
    // int count = 0;

    // Consider all possible pairs
    // and check their sums
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] + arr[j] == sum)
                System.out.println( "(" + arr[i] +
                                    ", " + arr[j] +
                                              ")");
}


// Driver Code
public static void main(String []arg)
```

```
{
    int arr[] = {1, 5, 7, -1, 5};
    int n = arr.length;
    int sum = 6;
    printPairs(arr, n, sum);
}
}

// This code is contributed
// by Smitha
```

Python 3

```
 # Python 3 implementation
# of simple method to find
# print pairs with given sum.

# Returns number of pairs
# in arr[0..n-1] with sum
# equal to 'sum'
def printPairs(arr, n, sum):

    # count = 0

    # Consider all possible
    # pairs and check their sums
    for i in range(0, n ):
        for j in range(i + 1, n ):
            if (arr[i] + arr[j] == sum):
                print("(" , arr[i] ,
                      ", ", arr[j],
                      ")", sep = "")


# Driver Code
arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6
printPairs(arr, n, sum)

# This code is contributed
# by Smitha
```

C#

```
 // C# implementation of simple
// method to find print pairs
```

```
// with given sum.
using System;

class GFG
{
// Returns number of pairs
// in arr[0..n-1] with sum
// equal to 'sum'
static void printPairs(int []arr,
                       int n, int sum)
{
    // int count = 0;

    // Consider all possible pairs
    // and check their sums
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] + arr[j] == sum)
                Console.Write( "(" + arr[i] +
                               ", " + arr[j] +
                               ")" + "\n");
}


// Driver Code
public static void Main()
{
    int []arr = { 1, 5, 7, -1, 5 };
    int n = arr.Length;
    int sum = 6;
    printPairs(arr, n, sum);
}
}

// This code is contributed
// by Smitha
```

PHP

```
 <?php
// PHP implementation of simple
// method to find print pairs
// with given sum.

// Returns number of pairs in
// arr[0..n-1] with sum equal
// to 'sum'
function printPairs($arr, $n, $sum)
```

```
{
    // Initialize result
    $count = 0;

    // Consider all possible
    // pairs and check their sums
    for ($i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)
            if ($arr[$i] + $arr[$j] == $sum)
                echo "(" , $arr[$i] , ", ",
                           $arr[$j] , ")" ,"\n";
}

// Driver Code
$arr = array (1, 5, 7, -1, 5);
$n = sizeof($arr);
$sum = 6;
printPairs($arr, $n, $sum);

// This code is contributed by m_kit
?>
```

Output :

```
(1, 5)
(1, 5)
(7, -1)
```

Method 2 (Use hashing).
We create an empty hash table. Now we traverse through the array and check for pairs in hash table. If a matching element is found, we print the pair number of times equal to number of occurrences of the matching element.

Note that the worst case of time complexity of this solution is **O(c + n)** where c is count of pairs with given sum.

C++

```
 // C++ implementation of simple method to
// find count of pairs with given sum.
#include <bits/stdc++.h>
using namespace std;

// Returns number of pairs in arr[0..n-1]
// with sum equal to 'sum'
void printPairs(int arr[], int n, int sum)
{
```

```
    // Store counts of all elements in map m
    unordered_map<int, int> m;

    // Traverse through all elements
    for (int i = 0; i < n; i++) {

        // Search if a pair can be formed with
        // arr[i].
        int rem = sum - arr[i];
        if (m.find(rem) != m.end()) {
            int count = m[rem];
            for (int j = 0; j < count; j++)
                cout << "(" << rem << ", "
                     << arr[i] << ")" << endl;
        }
        m[arr[i]]++;
    }
}

// Driver function to test the above function
int main()
{
    int arr[] = { 1, 5, 7, -1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int sum = 6;
    printPairs(arr, n, sum);
    return 0;
}
```

Output :

```
(1, 5)
(7, -1)
(1, 5)
```

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/print-all-pairs-with-given-sum/

Chapter 250

Print all subarrays with 0 sum

Print all subarrays with 0 sum - GeeksforGeeks

Given an array, print all subarrays in the array which has sum 0.

Examples:

```
Input:  arr = [6, 3, -1, -3, 4, -2, 2,
             4, 6, -12, -7]
Output:
Subarray found from Index 2 to 4
Subarray found from Index 2 to 6
Subarray found from Index 5 to 6
Subarray found from Index 6 to 9
Subarray found from Index 0 to 10
```

Related posts: Find if there is a subarray with 0 sum

A simple solution is to consider all subarrays one by one and check if sum of every subarray is equal to 0 or not. The complexity of this solution would be O(n^2).

A better approach is to use Hashing.

Do following for each element in the array

1. Maintain sum of elements encountered so far in a variable (say sum).
2. If current sum is 0, we found a subarray starting from index 0 and ending at index current index
3. Check if current sum exists in the hash table or not.
4. If current sum exists in the hash table, that means we have subarray(s) present with 0 sum that ends at current index.
5. Insert current sum into the hash table

Below is C++ implementation of above idea –

```
 // C++ program to print all subarrays
// in the array which has sum 0
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

// Function to print all subarrays in the array which
// has sum 0
vector< pair<int, int> > findSubArrays(int arr[], int n)
{
    // create an empty map
    unordered_map<int, vector<int> > map;

    // create an empty vector of pairs to store
    // subarray starting and ending index
    vector <pair<int, int>> out;

    // Maintains sum of elements so far
    int sum = 0;

    for (int i = 0; i < n; i++)
    {
        // add current element to sum
        sum += arr[i];

        // if sum is 0, we found a subarray starting
        // from index 0 and ending at index i
        if (sum == 0)
            out.push_back(make_pair(0, i));

        // If sum already exists in the map there exists
        // at-least one subarray ending at index i with
        // 0 sum
        if (map.find(sum) != map.end())
        {
            // map[sum] stores starting index of all subarrays
            vector<int> vc = map[sum];
            for (auto it = vc.begin(); it != vc.end(); it++)
                out.push_back(make_pair(*it + 1, i));
        }

        // Important - no else
        map[sum].push_back(i);
    }

    // return output vector
    return out;
```

```
}

// Utility function to print all subarrays with sum 0
void print(vector<pair<int, int>> out)
{
    for (auto it = out.begin(); it != out.end(); it++)
        cout << "Subarray found from Index " <<
            it->first << " to " << it->second << endl;
}

// Driver code
int main()
{
    int arr[] = {6, 3, -1, -3, 4, -2, 2, 4, 6, -12, -7};
    int n = sizeof(arr)/sizeof(arr[0]);

    vector<pair<int, int> > out = findSubArrays(arr, n);

    // if we didn't find any subarray with 0 sum,
    // then subarray doesn't exists
    if (out.size() == 0)
        cout << "No subarray exists";
    else
        print(out);

    return 0;
}
```

Output:

```
Subarray found from Index 2 to 4
Subarray found from Index 2 to 6
Subarray found from Index 5 to 6
Subarray found from Index 6 to 9
Subarray found from Index 0 to 10
```

Source

https://www.geeksforgeeks.org/print-all-subarrays-with-0-sum/

Chapter 251

Print all triplets in sorted array that form AP

Print all triplets in sorted array that form AP - GeeksforGeeks

Given a sorted array of distinct positive integers, print all triplets that form AP (or Arithmetic Progression)

Examples :

```
Input : arr[] = { 2, 6, 9, 12, 17, 22, 31, 32, 35, 42 };
Output :
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42

Input : arr[] = { 3, 5, 6, 7, 8, 10, 12};
Output :
3 5 7
5 6 7
6 7 8
6 8 10
8 10 12
```

A **simple solution** is to run three nested loops to generate all triplets and for every triplet, check if it forms AP or not. Time complexity of this solution is $O(n^3)$

A **better solution** is to use hashing. We traverse array from left to right. We consider every element as middle and all elements after it as next element. To search the previous element, we use hash table.

C++

```
 // C++ program to print all triplets in given
// array that form Arithmetic Progression
// C++ program to print all triplets in given
// array that form Arithmetic Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print all triplets in
// given sorted array that forms AP
void printAllAPTriplets(int arr[], int n)
{
    unordered_set<int> s;
    for (int i = 0; i < n - 1; i++)
    {
    for (int j = i + 1; j < n; j++)
    {
        // Use hash to find if there is
        // a previous element with difference
        // equal to arr[j] - arr[i]
        int diff = arr[j] - arr[i];
        if (s.find(arr[i] - diff) != s.end())
            cout << arr[i] - diff << " " << arr[i]
                 << " " << arr[j] << endl;
    }
    s.insert(arr[i]);
    }
}

// Driver code
int main()
{
    int arr[] = { 2, 6, 9, 12, 17,
                 22, 31, 32, 35, 42 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printAllAPTriplets(arr, n);
    return 0;
}
```

Java

```
 // Java program to print all
// triplets in given array
```

```
// that form Arithmetic
// Progression
import java.io.*;
import java.util.*;

class GFG
{
    // Function to print
    // all triplets in
    // given sorted array
    // that forms AP
    static void printAllAPTriplets(int []arr,
                                   int n)
    {
        ArrayList<Integer> s =
                 new ArrayList<Integer>();
        for (int i = 0;
                 i < n - 1; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                // Use hash to find if
                // there is a previous
                // element with difference
                // equal to arr[j] - arr[i]
                int diff = arr[j] - arr[i];
                boolean exists =
                        s.contains(arr[i] - diff);

                if (exists)
                    System.out.println(arr[i] - diff +
                                       " " + arr[i] +
                                       " " + arr[j]);
            }

        s.add(arr[i]);
        }
    }

    // Driver code
    public static void main(String args[])
    {
        int []arr = {2, 6, 9, 12, 17,
                     22, 31, 32, 35, 42};
        int n = arr.length;
        printAllAPTriplets(arr, n);
    }
}
```

```
// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python program to prall
# triplets in given array
# that form Arithmetic
# Progression

# Function to print
# all triplets in
# given sorted array
# that forms AP
def printAllAPTriplets(arr, n) :

    s = [];
    for i in range(0, n - 1) :

        for j in range(i + 1, n) :

            # Use hash to find if
            # there is a previous
            # element with difference
            # equal to arr[j] - arr[i]
            diff = arr[j] - arr[i];

            if ((arr[i] - diff) in arr) :
                print ("{} {} {}" .
                        format((arr[i] - diff),
                                arr[i], arr[j]),
                                    end = "\n");

    s.append(arr[i]);

# Driver code
arr = [2, 6, 9, 12, 17,
       22, 31, 32, 35, 42];
n = len(arr);
printAllAPTriplets(arr, n);

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to print all
```

```
// triplets in given array
// that form Arithmetic
// Progression
using System;
using System.Collections.Generic;

class GFG
{
    // Function to print
    // all triplets in
    // given sorted array
    // that forms AP
    static void printAllAPTriplets(int []arr,
                                   int n)
    {
        List<int> s = new List<int>();
        for (int i = 0;
                 i < n - 1; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                // Use hash to find if
                // there is a previous
                // element with difference
                // equal to arr[j] - arr[i]
                int diff = arr[j] - arr[i];
                bool exists = s.Exists(element =>
                                       element == (arr[i] -
                                                   diff));
                if (exists)
                    Console.WriteLine(arr[i] - diff +
                                      " " + arr[i] +
                                      " " + arr[j]);
            }
        s.Add(arr[i]);
        }
    }

    // Driver code
    static void Main()
    {
        int []arr = new int[]{ 2, 6, 9, 12, 17,
                               22, 31, 32, 35, 42 };
        int n = arr.Length;
        printAllAPTriplets(arr, n);
    }
}
// This code is contributed by
```

```
// Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP program to pr$all
// triplets in given array
// that form Arithmetic
// Progression

// Function to print
// all triplets in
// given sorted array
// that forms AP
function printAllAPTriplets($arr, $n)
{
    $s = array();
    for ($i = 0; $i < $n - 1; $i++)
    {
        for ($j = $i + 1;
             $j < $n; $j++)
        {
            // Use hash to find if
            // there is a previous
            // element with difference
            // equal to arr[j] - arr[i]
            $diff = $arr[$j] - $arr[$i];

            if (in_array($arr[$i] -
                        $diff, $arr))
                echo(($arr[$i] - $diff) .
                        " " . $arr[$i] .
                        " " . $arr[$j] . "\n");
        }
    array_push($s, $arr[$i]);
    }
}

// Driver code
$arr = array(2, 6, 9, 12, 17,
            22, 31, 32, 35, 42);
$n = count($arr);
printAllAPTriplets($arr, $n);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42
```

Time Complexity : O(n^2)
Auxiliary Space : O(n)

An **efficient solution** is based on the fact that array is sorted. We use the same concept as discussed in GP triplet question. The idea is to start from the second element and fix every element as middle element and search for the other two elements in a triplet (one smaller and one greater).

Below is the implementation of the above idea.

C++

```
 // C++ program to print all triplets in given
// array that form Arithmetic Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print all triplets in
// given sorted array that forms AP
void printAllAPTriplets(int arr[], int n)
{
    for (int i = 1; i < n - 1; i++)
    {

        // Search other two elements of
        // AP with arr[i] as middle.
        for (int j = i - 1, k = i + 1; j >= 0 && k < n;)
        {

            // if a triplet is found
            if (arr[j] + arr[k] == 2 * arr[i])
            {
                cout << arr[j] << " " << arr[i]
                     << " " << arr[k] << endl;

                // Since elements are distinct,
```

```
                // arr[k] and arr[j] cannot form
                // any more triplets with arr[i]
                k++;
                j--;
            }

            // If middle element is more move to
            // higher side, else move lower side.
            else if (arr[j] + arr[k] < 2 * arr[i])
                k++;
            else
                j--;
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 2, 6, 9, 12, 17,
                  22, 31, 32, 35, 42 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printAllAPTriplets(arr, n);
    return 0;
}
```

Java

```
 // Java implementation to print
// all the triplets in given array
// that form Arithmetic Progression

import java.io.*;

class GFG
{

    // Function to print all triplets in
    // given sorted array that forms AP
    static void findAllTriplets(int arr[], int n)
    {

        for (int i = 1; i < n - 1; i++)
        {

            // Search other two elements
            // of AP with arr[i] as middle.
            for (int j = i - 1, k = i + 1; j >= 0 && k < n;)
```

```
            {

                // if a triplet is found
                if (arr[j] + arr[k] == 2 * arr[i])
                {
                    System.out.println(arr[j] +" " +
                                       arr[i]+ " " + arr[k]);

                    // Since elements are distinct,
                    // arr[k] and arr[j] cannot form
                    // any more triplets with arr[i]
                    k++;
                    j--;
                }

                // If middle element is more move to
                // higher side, else move lower side.
                else if (arr[j] + arr[k] < 2 * arr[i])
                    k++;
                else
                    j--;
            }
        }
    }

    // Driver code
    public static void main (String[] args)
    {

        int arr[] = { 2, 6, 9, 12, 17,
                      22, 31, 32, 35, 42 };
        int n = arr.length;

        findAllTriplets(arr, n);
    }
}

// This code is contributed by vt_m.
```

Python 3

```
 # python 3 program to print all triplets in given
# array that form Arithmetic Progression

# Function to print all triplets in
# given sorted array that forms AP
def printAllAPTriplets(arr, n):
```

```
    for i in range(1, n - 1):

        # Search other two elements of
        # AP with arr[i] as middle.
        j = i - 1
        k = i + 1
        while(j >= 0 and k < n ):

            # if a triplet is found
            if (arr[j] + arr[k] == 2 * arr[i]):
                print(arr[j], "", arr[i], "", arr[k])

                # Since elements are distinct,
                # arr[k] and arr[j] cannot form
                # any more triplets with arr[i]
                k += 1
                j -= 1


            # If middle element is more move to
            # higher side, else move lower side.
            elif (arr[j] + arr[k] < 2 * arr[i]):
                k += 1
            else:
                j -= 1

# Driver code
arr = [ 2, 6, 9, 12, 17,
        22, 31, 32, 35, 42 ]
n = len(arr)
printAllAPTriplets(arr, n)

# This article is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# implementation to print
// all the triplets in given array
// that form Arithmetic Progression

using System;

class GFG
{

    // Function to print all triplets in
    // given sorted array that forms AP
```

```
    static void findAllTriplets(int []arr, int n)
    {

        for (int i = 1; i < n - 1; i++)
        {

            // Search other two elements
            // of AP with arr[i] as middle.
            for (int j = i - 1, k = i + 1; j >= 0 && k < n;)
            {

                // if a triplet is found
                if (arr[j] + arr[k] == 2 * arr[i])
                {
                    Console.WriteLine(arr[j] +" " +
                                      arr[i]+ " " + arr[k]);

                    // Since elements are distinct,
                    // arr[k] and arr[j] cannot form
                    // any more triplets with arr[i]
                    k++;
                    j--;
                }

                // If middle element is more move to
                // higher side, else move lower side.
                else if (arr[j] + arr[k] < 2 * arr[i])
                    k++;
                else
                    j--;
            }
        }
    }

    // Driver code
    public static void Main ()
    {

        int []arr = { 2, 6, 9, 12, 17,
                      22, 31, 32, 35, 42 };
        int n = arr.Length;

        findAllTriplets(arr, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP implementation to print
// all the triplets in given array
// that form Arithmetic Progression

// Function to print all triplets in
// given sorted array that forms AP
function findAllTriplets($arr, $n)
{
    for ($i = 1; $i < $n - 1; $i++)
    {

        // Search other two elements
        // of AP with arr[i] as middle.
        for ($j = $i - 1, $k = $i + 1;
                  $j >= 0 && $k < $n
        {

            // if a triplet is found
            if ($arr[$j] + $arr[$k] == 2 *
                           $arr[$i])
            {
                echo $arr[$j] ." " .
                     $arr[$i]. " " .
                     $arr[$k] . "\n";

                // Since elements are distinct,
                // arr[k] and arr[j] cannot form
                // any more triplets with arr[i]
                $k++;
                $j--;
            }

            // If middle element is more move to
            // higher side, else move lower side.
            else if ($arr[$j] + $arr[$k] < 2 *
                                    $arr[$i])
                $k++;
            else
                $j--;
        }
    }
}

// Driver code
$arr = array(2, 6, 9, 12, 17,
```

```
            22, 31, 32, 35, 42);

$n = count($arr);
findAllTriplets($arr, $n);

// This code is contributed by Sam007
?>
```

Output :

```
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42
```

Time Complexity : O(n^2)
Auxiliary Space : O(1)

Improved By : nitin mittal, Sam007, manishshaw1

Source

https://www.geeksforgeeks.org/print-triplets-sorted-array-form-ap/

Chapter 252

Print all triplets with given sum

Print all triplets with given sum - GeeksforGeeks

Given an array of distinct elements. The task is to find triplets in array whose sum is equal to a given number.

Examples :

```
Input : arr[] = {0, -1, 2, -3, 1}
        sum = -2
Output : 0  -3  1
         -1  2 -3

Input : arr[] = {1, -2, 1, 0, 5}
        sum = 0
Output : 1 -2  1
```

Method 1 (Simple : $O(n^3)$)
The naive approach is that run three loops and check one by one that sum of three elements is given sum or not If sum of three elements is given sum, then print elements other wise print not found.

C++

```
 // A simple C++ program to find three elements
// whose sum is equal to given sum
#include <bits/stdc++.h>
using namespace std;

// Prints all triplets in arr[] with given sum
void findTriplets(int arr[], int n, int sum)
{
```

```
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            for (int k = j + 1; k < n; k++) {
                if (arr[i] + arr[j] + arr[k] == sum) {
                    cout << arr[i] << " "
                         << arr[j] << " "
                         << arr[k] << endl;
                }
            }
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, -2);
    return 0;
}
```

Java

```
 // A simple Java program
// to find three elements
// whose sum is equal to
// given sum
import java.io.*;

class GFG
{

// Prints all triplets in
// arr[] with given sum
static void findTriplets(int arr[],
                         int n, int sum)
{
    for (int i = 0;
             i < n - 2; i++)
    {
        for (int j = i + 1;
                 j < n - 1; j++)
        {
            for (int k = j + 1;
                     k < n; k++)
            {
                if (arr[i] + arr[j] + arr[k] == sum)
```

```
                {
                    System.out.println(arr[i]+ " "+
                                       arr[j] +" "+
                                       arr[k] );
                }
            }
        }
    }
}

// Driver code
public static void main (String[] args)
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = arr.length;
    findTriplets(arr, n, -2);
}
}

// This code is contributed by m_kit
```

Python 3

```
 # A simple Python 3 program
# to find three elements
# whose sum is equal to
# given sum

# Prints all triplets in
# arr[] with given sum
def findTriplets(arr, n, sum):

    for i in range(0 , n - 2):
        for j in range(i + 1 , n - 1):
            for k in range(j + 1, n):
                if (arr[i] + arr[j] +
                    arr[k] == sum):
                    print(arr[i], " ",
                          arr[j] ," ",
                          arr[k] , sep = "")

# Driver code
arr = [ 0, -1, 2, -3, 1 ]
n = len(arr)
findTriplets(arr, n, -2)

# This code is contributed
# by Smitha
```

C#

```
 // A simple C# program
// to find three elements
// whose sum is equal to
// given sum
using System;

class GFG
{

// Prints all triplets in
// arr[] with given sum
static void findTriplets(int []arr,
                         int n, int sum)
{
    for (int i = 0;
             i < n - 2; i++)
    {
        for (int j = i + 1;
                 j < n - 1; j++)
        {
            for (int k = j + 1;
                     k < n; k++)
            {
                if (arr[i] + arr[j] + arr[k] == sum)
                {
                    Console.WriteLine(arr[i]+ " "+
                                      arr[j] +" "+
                                      arr[k] );
                }
            }
        }
    }
}

// Driver code
static public void Main ()
{
    int []arr = {0, -1, 2, -3, 1};
    int n = arr.Length;
    findTriplets(arr, n, -2);
}
}

// This code is contributed by akt_mit
```

PHP

```
 <?php
// A simple PHP program to
// find three elements whose
// sum is equal to given sum

// Prints all triplets in
// arr[] with given sum
function findTriplets($arr, $n, $sum)
{
    for ($i = 0; $i < $n - 2; $i++)
    {
        for ($j = $i + 1; $j < $n - 1; $j++)
        {
            for ($k = $j + 1; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] +
                    $arr[$k] == $sum)
                {
                    echo $arr[$i] , " ",
                         $arr[$j] , " ",
                         $arr[$k] , "\n";
                }
            }
        }
    }
}

// Driver code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n, -2);

// This code is contributed by aj_36
?>
```

Output :

```
0 -3 1
-1 2 -3
```

Time Complexity : $O(n^3)$
Auxiliary Space : $O(1)$

Method 2 (Hashing : $O(n^2)$)
We iterate through every element. For every element arr[i], we find a pair with sum "-arr[i]". This problem reduces to pairs sum and can be solved in O(n) time using hashing.

```
Run a loop from i=0 to n-2
  Create an empty hash table
  Run inner loop from j=i+1 to n-1
      If -(arr[i] + arr[j]) is present in hash table
         print arr[i], arr[j] and -(arr[i]+arr[j])
      Else
         Insert arr[j] in hash table.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is equal to given sum.
#include <bits/stdc++.h>
using namespace std;

// function to print triplets with given sum
void findTriplets(int arr[], int n, int sum)
{
    for (int i = 0; i < n - 1; i++) {
        // Find all pairs with sum equals to
        // "sum-arr[i]"
        unordered_set<int> s;
        for (int j = i + 1; j < n; j++) {
            int x = sum - (arr[i] + arr[j]);
            if (s.find(x) != s.end())
                printf("%d %d %d\n", x, arr[i], arr[j]);
            else
                s.insert(arr[j]);
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, sum);
    return 0;
}
```

Output:

```
-3 0 1
2 -1 -3
```

Time Complexity : $O(n^2)$
Auxiliary Space : O(n)

Method 3 (Sorting : $O(n^2)$)
The above method requires extra space. We can solve in O(1) extra space. The idea is based on method 2 of this post.

```
1. Sort all element of array
2. Run loop from i=0 to n-2.
     Initialize two index variables l=i+1 and r=n-1
4. while (l < r)
     Check sum of arr[i], arr[l], arr[r] is
     given sum or not if sum is 'sum', then print
     the triplet and do l++ and r--.
5. If sum is less than given sum then l++
6. If sum is greater than given sum then r--
7. If not exist in array then print not found.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is given sum.
#include <bits/stdc++.h>
using namespace std;

// function to print triplets with given sum
void findTriplets(int arr[], int n, int sum)
{
    // sort array elements
    sort(arr, arr + n);

    for (int i = 0; i < n - 1; i++) {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r) {
            if (x + arr[l] + arr[r] == sum) {
                // print elements if it's sum is given sum.
                printf("%d %d %d\n", x, arr[l], arr[r]);
                l++;
                r--;
            }

            // If sum of three elements is less
            // than 'sum' then increment in left
            else if (x + arr[l] + arr[r] < sum)
```

```
                l++;

            // if sum is greater than given sum, then
            // decrement in right side
            else
                r--;
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, sum);
    return 0;
}
```

Java

```
 // Java program to find triplets
// in a given array whose sum
// is given sum.
import java.io.*;
import java.util.*;

class GFG
{

// function to print
// triplets with given sum
static void findTriplets(int[] arr,
                         int n, int sum)
{
    // sort array elements
    Arrays.sort(arr);

    for (int i = 0;
             i < n - 1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
```

```
            if (x + arr[l] + arr[r] == sum)
            {
                // print elements if it's
                // sum is given sum.
                System.out.println(x + " " + arr[l] +
                                        " " + arr[r]);
                l++;
                r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if (x + arr[l] +
                         arr[r] < sum)
                l++;

            // if sum is greater than
            // given sum, then decrement
            // in right side
            else
                r--;
        }
    }
}

// Driver code
public static void main(String args[])
{
    int[] arr = new int[]{ 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = arr.length;
    findTriplets(arr, n, sum);
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

C#

```
 // C# program to find triplets
// in a given array whose sum
// is given sum.
using System;

class GFG
{
```

```
// function to print
// triplets with given sum
static void findTriplets(int[] arr,
                         int n, int sum)
{
    // sort array elements
    Array.Sort(arr);

    for (int i = 0; i < n - 1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
            if (x + arr[l] + arr[r] == sum)
            {
                // print elements if it's
                // sum is given sum.
                Console.WriteLine(x + " " + arr[l] +
                                  " " + arr[r]);
                l++;
                r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if (x + arr[l] +
                         arr[r] < sum)
                l++;

            // if sum is greater than
            // given sum, then decrement
            // in right side
            else
                r--;
        }
    }
}

// Driver code
static int Main()
{
    int[] arr = new int[]{ 0, -1, 2, -3, 1 };
    int sum = -2;
```

```
    int n = arr.Length;
    findTriplets(arr, n, sum);
    return 0;
}
}

// This code is contributed by rahul
```

PHP

```
 <?php
// PHP program to find triplets
// in a given array whose sum
// is given sum.

// function to print triplets
// with given sum
function findTriplets($arr, $n, $sum)
{
    // sort array elements
    sort($arr);

    for ($i = 0; $i < $n - 1; $i++)
    {
        // initialize left and right
        $l = $i + 1;
        $r = $n - 1;
        $x = $arr[$i];
        while ($l < $r)
        {
            if ($x + $arr[$l] +
                $arr[$r] == $sum)
            {
                // print elements if it's
                // sum is given sum.
                echo $x, " ", $arr[$l],
                         " ", $arr[$r], "\n";
                $l++;
                $r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if ($x + $arr[$l] +
                     $arr[$r] < $sum)
                $l++;
```

```
            // if sum is greater
            // than given sum, then
            // decrement in right side
            else
                $r--;
        }
    }
}

// Driver code
$arr = array(0, -1, 2, -3, 1);
$sum = -2;
$n = sizeof($arr);
findTriplets($arr, $n, $sum);

// This code is contributed by ajit
?>
```

Output:

```
-3 -1 2
-3 0 1
```

Time Complexity : O(n^2)
Auxiliary Space : O(1)

Improved By : jit_t, Smitha Dinesh Semwal, mithunkumarmnnit321, Abby_akku

Source

https://www.geeksforgeeks.org/print-all-triplets-with-given-sum/

Chapter 253

Print array elements that are divisible by at-least one other

Print array elements that are divisible by at-least one other - GeeksforGeeks

Given an array of length **N** that contains only integers, the task is to print the special numbers of array. A number in this array is called **Special** number if it is divisible by at least one other number in the array.

Examples :

Input : 1 2 3
Output : 2 3
Explanation : both 2 and 3 are divisible by 1.

Input : 2 3 4 6 8 9
Output : 4 6 8 9
Explanation : 2 and 3 are not divisible by any other element. Rest of the element are divisible by at-least 1 element. 6 is divisible by both 2 and 3, 4 divisible by 2, 8 divisible by 2 and 4 both, 9 divisible by 3.

Input : 3 5 7 11
Output :
Explanation : all elements are relatively prime so no special number.

A **simple solution** is to traverse through all elements, then check for every element if it is divisible by any other. Time complexity of this solution is $O(n^2)$

Another solution that works better when there are many elements with not very big values. Store all array elements into hash and find out the max element in array then up-to max element find out the multiples of a given number then if multiple of array element is in hash then that number is divisible by at-least one element of array .To remove duplicate values we store the value into set because if array has 2, 3 and 6 then only 6 is divisible by at-least one element of array, both 2 and 3 divide 6 so 6 will be stored only one

time.

C++

```
 // C++ program to find special numbers
// in an array
#include <bits/stdc++.h>
using namespace std;

// Function to find special numbers
void divisibilityCheck(int arr[], int n)
{
    // Storing all array elements in a hash
    // and finding maximum element in array
    unordered_set<int> s;
    int max_ele = INT_MIN;
    for (int i = 0; i < n; i++) {
        s.insert(arr[i]);

        // finding maximum element of array
        max_ele = max(max_ele, arr[i]);
    }

    // traversing array element and storing the array
    // multiples that are present in s in res.
    unordered_set<int> res;
    for (int i = 0; i < n; i++) {

        // Check for non-zero values only
        if (arr[i] != 0)

            // checking the factor of current element
            for (int j = arr[i] * 2; j <= max_ele;
                                        j += arr[i])
             {

                // if factor is already part of array
                // element then store it
                if (s.find(j) != s.end())
                    res.insert(j);
            }
    }

    // displaying elements that are divisible by
    // at least one other in array
    for (auto x : res)
        cout << x << " ";
}
```

```
// Driver code
int main()
{
    int arr[] = { 2, 3, 8, 6, 9, 10 };
    int n = sizeof(arr) / sizeof(arr[0]);
    divisibilityCheck(arr, n);
    return 0;
}
```

Java

```
 // Java program to find special
// numbers in an array
import java.io.*;
import java.util.*;

class GFG
{
    // Function to find
    // special numbers
    static void divisibilityCheck(List<Integer> arr,
                                                int n)
    {
        // Storing all array elements
        // in a hash and finding maximum
        // element in array
        List<Integer> s = new ArrayList<Integer>();
        int max_ele = Integer.MIN_VALUE;
        for (int i = 0; i < n; i++)
        {
            s.add(arr.get(i));

            // finding maximum
            // element of array
            max_ele = Math.max(max_ele,
                               arr.get(i));
        }

        // traversing array element and
        // storing the array multiples
        // that are present in s in res.
        LinkedHashSet<Integer> res =
                        new LinkedHashSet<Integer>();
        for (int i = 0; i < n; i++)
        {

            // Check for non-zero values only
```

```
            if (arr.get(i) != 0)

                // checking the factor
                // of current element
                for (int j = arr.get(i) * 2;
                         j <= max_ele;
                         j += arr.get(i))
                {

                    // if factor is already
                    // part of array element
                    // then store it
                    if (s.contains(j))
                        res.add(j);
                }
        }

        // displaying elements that
        // are divisible by at least
        // one other in array
        List<Integer> list =
                 new ArrayList<Integer>(res);
        Collections.reverse(list);

        for (Integer temp : list)
        System.out.print(temp + " ");
    }

    // Driver Code
    public static void main(String args[])
    {
        List<Integer> arr = Arrays.asList(2, 3, 8,
                                          6, 9, 10);
        int n = arr.size();
        divisibilityCheck(arr, n);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find special
// numbers in an array
using System;
using System.Linq;
using System.Collections.Generic;
```

```
class GFG
{
    // Function to find
    // special numbers
    static void divisibilityCheck(List<int> arr,
                                      int n)
    {
        // Storing all array elements
        // in a hash and finding maximum
        // element in array
        List<int> s = new List<int>();
        int max_ele = Int32.MinValue;
        for (int i = 0; i < n; i++)
        {
            s.Add(arr[i]);

            // finding maximum element of array
            max_ele = Math.Max(max_ele,
                               arr[i]);
        }

        // traversing array element and
        // storing the array multiples
        // that are present in s in res.
        HashSet<int> res = new HashSet<int>();
        for (int i = 0; i < n; i++)
        {

            // Check for non-zero values only
            if (arr[i] != 0)

                // checking the factor
                // of current element
                for (int j = arr[i] * 2; j <= max_ele;
                                         j += arr[i])
                {

                    // if factor is already part
                    // of array element then store it
                    if (s.Contains(j))
                        res.Add(j);
                }
        }
        // displaying elements that
        // are divisible by at least
        // one other in array
        foreach (int i in res.Reverse())
```

```
            Console.Write(i + " ");
    }

    // Driver Code
    static void Main()
    {
        List<int> arr = new List<int>(){ 2, 3, 8,
                                         6, 9, 10 };
        int n = arr.Count;
        divisibilityCheck(arr, n);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
9 10 8 6
```

Note : If we need result to be printed in sorted order, we can use set in place of unordered_set.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/divisibility-check/

Chapter 254

Print n smallest elements from given array in their original order

Print n smallest elements from given array in their original order - GeeksforGeeks

We are given an array of m-elements, we need to find n smallest elements from the array but they must be in the same order as they are in given array.

Examples:

```
Input : arr[] = {4, 2, 6, 1, 5},
        n = 3
Output : 4 2 1
Explanation :
1, 2 and 4 are 3 smallest numbers and
4 2 1 is their order in given array.

Input : arr[] = {4, 12, 16, 21, 25},
        n = 3
Output : 4 12 16
Explanation :
4, 12 and 16 are 3 smallest numbers and
4 12 16 is their order in given array.
```

Make a copy of original array and then sort copy array. After sorting the copy array, save all n smallest numbers. Further for each element in original array, check whether it is in n-smallest number or not if it present in n-smallest array then print it otherwise move forward.

Make copy_arr[]

sort(copy_arr)

For all elements in arr[] -

- Find arr[i] in n-smallest element of copy_arr
- If found then print the element

Below is CPP implementation of above approach :

```
 // CPP for printing smallest n number in order
#include <algorithm>
#include <iostream>
using namespace std;

// Function to print smallest n numbers
void printSmall(int arr[], int asize, int n)
{
    // Make copy of array
    vector<int> copy_arr(arr, arr + asize);

    // Sort copy array
    sort(copy_arr.begin(), copy_arr.begin() + asize);

    // For each arr[i] find whether
    // it is a part of n-smallest
    // with binary search
    for (int i = 0; i < asize; ++i)
        if (binary_search(copy_arr.begin(),
                copy_arr.begin() + n, arr[i]))
            cout << arr[i] << " ";
}

// Driver program
int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int asize = sizeof(arr) / sizeof(arr[0]);
    int n = 5;
    printSmall(arr, asize, n);
    return 0;
}
```

Output :

```
1 3 4 2 0
```

For making a copy of array we need space complexity of O(n) and then for sorting we will need complexity of order O(n log n). Further for each element in arr[] we are performing searching in copy_arr[], which will result O(n) for linear search but we can improve it by applying binary search and hence our overall time complexity will be **O(n log n)**.

Source

https://www.geeksforgeeks.org/find-n-smallest-element-given-array-order-array/

Chapter 255

Print the last occurrence of elements in array in relative order

Print the last occurrence of elements in array in relative order - GeeksforGeeks

Given an array of N elements, print the elements in the same relative order as given by removing all the occurrences of elements except the last occurrence.

Examples:

> **Input:** a[] = {1, 5, 5, 1, 6, 1}
> **Output:** 5 6 1
> Remove two integers 1, which are in the positions 1 and 4. Also, remove the integer 5, which is in the position 2.
> Hence the left array is {5, 6, 1}
>
> **Input:** a[] = {2, 5, 5, 2}
> **Output:** 5 2

Approach:

- Hash the last occurrence of every element.
- Iterate in the array of N elements, if the element's index is hashed, then print the array element.

Below is the implementation of the above approach:

C++

```
 // C++ program to print the last occurrence
// of every element in relative order
#include <bits/stdc++.h>
using namespace std;

// Function to print the last occurrence
// of every element in an array
void printLastOccurrence(int a[], int n)
{

    // used in hashing
    unordered_map<int, int> mp;

    // iterate and store the last index
    // of every element
    for (int i = 0; i < n; i++)
        mp[a[i]] = i;

    // iterate and check for the last
    // occurrence of every element
    for (int i = 0; i < n; i++) {
        if (mp[a[i]] == i)
            cout << a[i] << " ";
    }
}
// Driver Code
int main()
{
    int a[] = { 1, 5, 5, 1, 6, 1 };
    int n = sizeof(a) / sizeof(a[0]);
    printLastOccurrence(a, n);

    return 0;
}
```

Java

```
 // Java program to print the
// last occurrence of every
// element in relative order
import java.util.*;

class GFG
{

    // Function to print the last
    // occurrence of every element
    // in an array
```

```
    public static void printLastOccurrence(int a[],
                                           int n)
    {
        HashMap<Integer,
                Integer> map = new HashMap<Integer,
                                           Integer>();

        // iterate and store the last
        // index of every element
        for (int i = 0; i < n; i++)
            map.put(a[i], i);

        for (int i = 0; i < n; i++)
        {
        if (map.get(a[i]) == i)
            System.out.print(a[i] +" ");
        }
    }

    // Driver Code
    public static void main (String[] args)
    {
        int a[] = { 1, 5, 5, 1, 6, 1 };
        int n = a.length;
        printLastOccurrence(a, n);
    }
}

// This code is contributed
// by ankita_saini
```

C#

```
// C# program to print the
// last occurrence of every
// element in relative order
using System;

class GFG
{

// Function to print the last
// occurrence of every element
// in an array
public static void printLastOccurrence(int[] a,
int n)
{
HashMap map = new HashMap();

// iterate and store the last
```

// index of every element
for (int i = 0; i < n; i++) map.put(a[i], i); for (int i = 0; i < n; i++) { if (map.get(a[i]) == i) Console.Write(a[i] + " "); } } // Driver Code public static void Main () { int[] a = { 1, 5, 5, 1, 6, 1 }; int n = a.Length; printLastOccurrence(a, n); } } // This code is contributed // by ChitraNayal [tabby title="PHP"]

Output:

```
5 6 1
```

Time Complexity: O(N * log N)

Improved By : ankita_saini, ChitraNayal

Source

https://www.geeksforgeeks.org/print-the-last-occurrence-of-elements-in-array-in-relative-order/

Chapter 256

Printing longest Increasing consecutive subsequence

Printing longest Increasing consecutive subsequence - GeeksforGeeks

Given n elements, write a program that prints the longest increasing subsequence whose adjacent element difference is one.

Examples:

> Input : a[] = {3, 10, 3, 11, 4, 5, 6, 7, 8, 12}
> Output : 3 4 5 6 7 8
> Explanation: 3, 4, 5, 6, 7, 8 is the longest increasing subsequence whose adjacent element differs by one.
>
> Input : a[] = {6, 7, 8, 3, 4, 5, 9, 10}
> Output : 6 7 8 9 10
> Explanation: 6, 7, 8, 9, 10 is the longest increasing subsequence

We have discussed how to find length of Longest Increasing consecutive subsequence. To print the subsequence, we store index of last element. Then we print consecutive elements ending with last element.

Given below is the implementation of the above approach:

```
 // CPP program to find length of the
// longest increasing subsequence
// whose adjacent element differ by 1
#include <bits/stdc++.h>
using namespace std;

// function that returns the length of the
// longest increasing subsequence
// whose adjacent element differ by 1
```

```
void longestSubsequence(int a[], int n)
{
    // stores the index of elements
    unordered_map<int, int> mp;

    // stores the length of the longest
    // subsequence that ends with a[i]
    int dp[n];
    memset(dp, 0, sizeof(dp));

    int maximum = INT_MIN;

    // iterate for all element
    int index = -1;
    for (int i = 0; i < n; i++) {

        // if a[i]-1 is present before i-th index
        if (mp.find(a[i] - 1) != mp.end()) {

            // last index of a[i]-1
            int lastIndex = mp[a[i] - 1] - 1;

            // relation
            dp[i] = 1 + dp[lastIndex];
        }
        else
            dp[i] = 1;

        // stores the index as 1-index as we need to
        // check for occurrence, hence 0-th index
        // will not be possible to check
        mp[a[i]] = i + 1;

        // stores the longest length
        if (maximum < dp[i]) {
            maximum = dp[i];
            index = i;
        }
    }

    // We know last element of sequence is
    // a[index]. We also know that length
    // of subsequence is "maximum". So We
    // print these many consecutive elements
    // starting from "a[index] - maximum + 1"
    // to a[index].
    for (int curr = a[index] - maximum + 1;
         curr <= a[index]; curr++)
```

```
        cout << curr << " ";
}

// Driver Code
int main()
{
    int a[] = { 3, 10, 3, 11, 4, 5, 6, 7, 8, 12 };
    int n = sizeof(a) / sizeof(a[0]);
    longestSubsequence(a, n);
    return 0;
}
```

Output:

```
3 4 5 6 7 8
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/printing-longest-increasing-consecutive-subsequence/

Chapter 257

Program to sort string in descending order

Program to sort string in descending order - GeeksforGeeks

Given a string, sort it in descending order.

Examples:

```
Input : alkasingh
Output : snlkihgaa

Input : nupursingh
Output : uusrpnnihg

Input : geeksforgeeks
Output : ssrokkggfeeee
```

A **simple solution** is to use library sort function std::sort()

C++

```
 // CPP program to sort a string in descending
// order using library function
#include <bits/stdc++.h>
using namespace std;

void descOrder(string s)
{
    sort(s.begin(), s.end(), greater<char>());
}
```

```
int main()
{
    string s = "geeksforgeeks";
    descOrder(s); // function call
    return 0;
}
```

Python

```
 # Python program to sort
# a string in descending
# order using library function

def descOrder(s):
    s.sort(reverse = True)
    str1 = ''.join(s)
    print(str1)

def main():
    s = list('geeksforgeeks')

    # function call
    descOrder(s)

if __name__=="__main__":
    main()

# This code is contributed by
# prabhat kumar singh
```

Output:

```
 ssrokkggfeeee
```

The time complexity is : O(n log n)

An **efficient approach** will be to observe first that there can be a total of 26 unique characters only. So, we can store the count of occurrences of all the characters from ‘a’ to ‘z’ in a hashed array. The first index of the hashed array will represent character ‘a’, second will represent ‘b’ and so on. Finally, we will simply traverse the hashed array and print the characters from ‘z’ to ‘a’ the number of times they occurred in input string.

Below is the implementation of above idea:

C++

```
 // C++ program to sort a string of characters
// in descending order
#include <bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

// function to print string in sorted order
void sortString(string& str)
{
    // Hash array to keep count of characters.
    // Initially count of all charters is
    // initialized to zero.
    int charCount[MAX_CHAR] = { 0 };

    // Traverse string and increment
    // count of characters
    for (int i = 0; i < str.length(); i++)

        // 'a'-'a' will be 0, 'b'-'a' will be 1,
        // so for location of character in count
        // array we wil do str[i]-'a'.
        charCount[str[i] - 'a']++;

    // Traverse the hash array and print
    // characters
    for (int i = MAX_CHAR - 1; i >= 0; i--)
        for (int j = 0; j < charCount[i]; j++)
            cout << (char)('a' + i);
}

// Driver program to test above function
int main()
{
    string s = "alkasingh";
    sortString(s);
    return 0;
}
```

Output:

```
snlkihgaa
```

Time Complexity: O(n), where n is the length of input string.
Auxiliary Space: O(1).

Improved By : prabhat kumar singh

Source

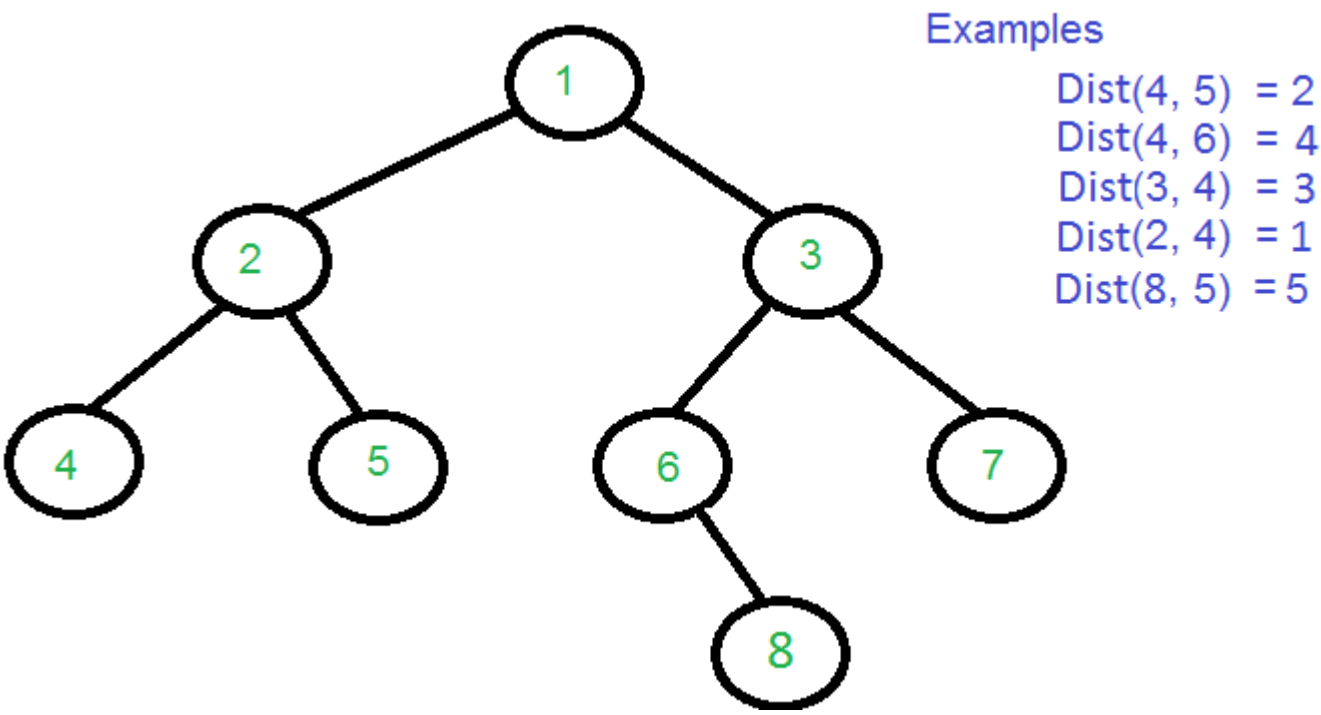
https://www.geeksforgeeks.org/program-sort-string-descending-order/

Chapter 258

Queries to find distance between two nodes of a Binary tree

Queries to find distance between two nodes of a Binary tree - GeeksforGeeks

Given a binary tree, the task is to find the distance between two keys in a binary tree, no parent pointers are given. The distance between two nodes is the minimum number of edges to be traversed to reach one node from other.
It has been already discussed in this for a single query in O(logn) time, here the task is to reduce multiple queries time to O(1) by compromising with space complexity to O(Nlogn).
In this post, we will use **Sparse table** instead of segment tree for finding the minimum in given range, which uses dynamic programming and bit manipulation to achieve O(1) query time.



A sparse table will preprocess the minimum values given for an array in Nlogn space i.e. each node will contain chain of values of log(i) length where i is the index of the ith node in L array. Each entry in the sparse table says M[i][j] will represent the index of the minimum value in the subarray starting at i having length 2^j.

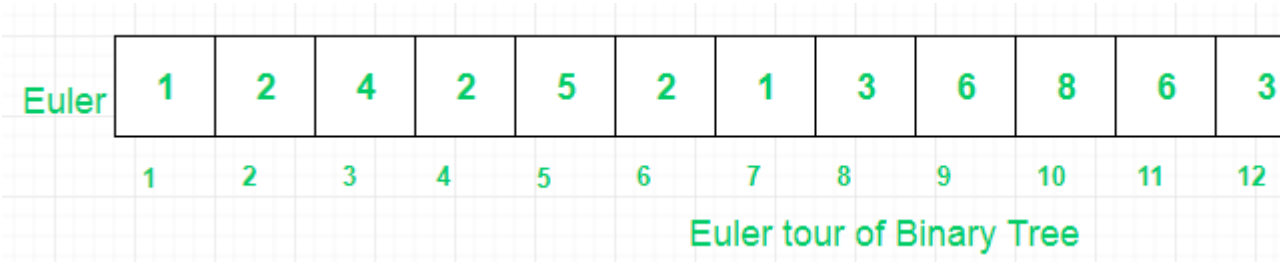
The distance between two nodes can be obtained in terms of lowest common ancestor.

```
Dist(n1, n2) = Level[n1] + Level[n2] - 2*Level[lca]
```

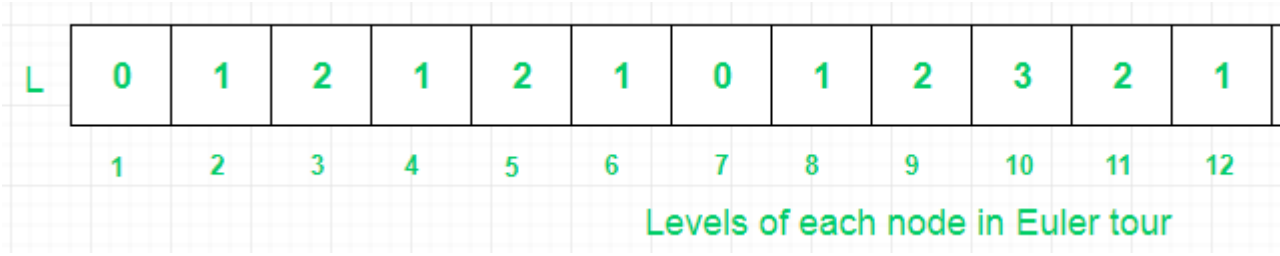
This problem can be breakdown into **finding levels of each node**, **finding the Euler tour** of binary tree and **building sparse table** for LCA, these steps are explained below :

1. Find the levels of each node by applying level order traversal.
2. Find the LCA of two nodes in the binary tree in O(logn) by Storing Euler tour of tree in array and computing two other arrays with the help of levels of each node and Euler tour.
   These steps are shown below:

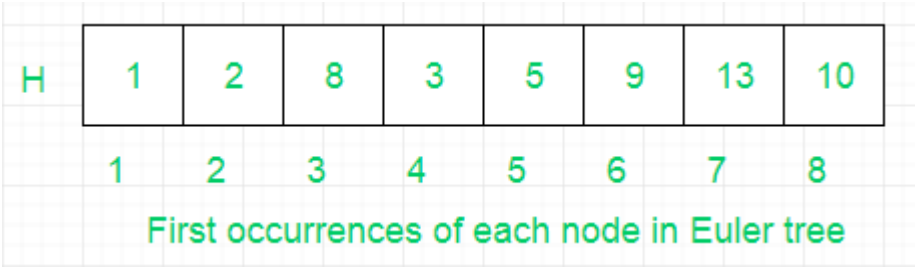
   (I) First, find Euler Tour of binary tree.



(II) Then, store levels of each node in Euler array.



(III) Then, store First occurrences of all nodes of binary tree in Euler array.



3. Then build sparse table on L array and find the minimum value say X in range (H[A] to H[B]). Then use the index of value X as an index to Euler array to get LCA, i.e. Euler[index(X)].

   Let, A=8 and B=5.
   (I) H[8]= 1 and H[5]=2
   (II) we get min value in L array between 1 and 2 as X=0, index=7
   (III) Then, LCA= Euler[7], i.e LCA=1.

4. Finally, apply distance formula discussed above to get the distance between two nodes.

Source

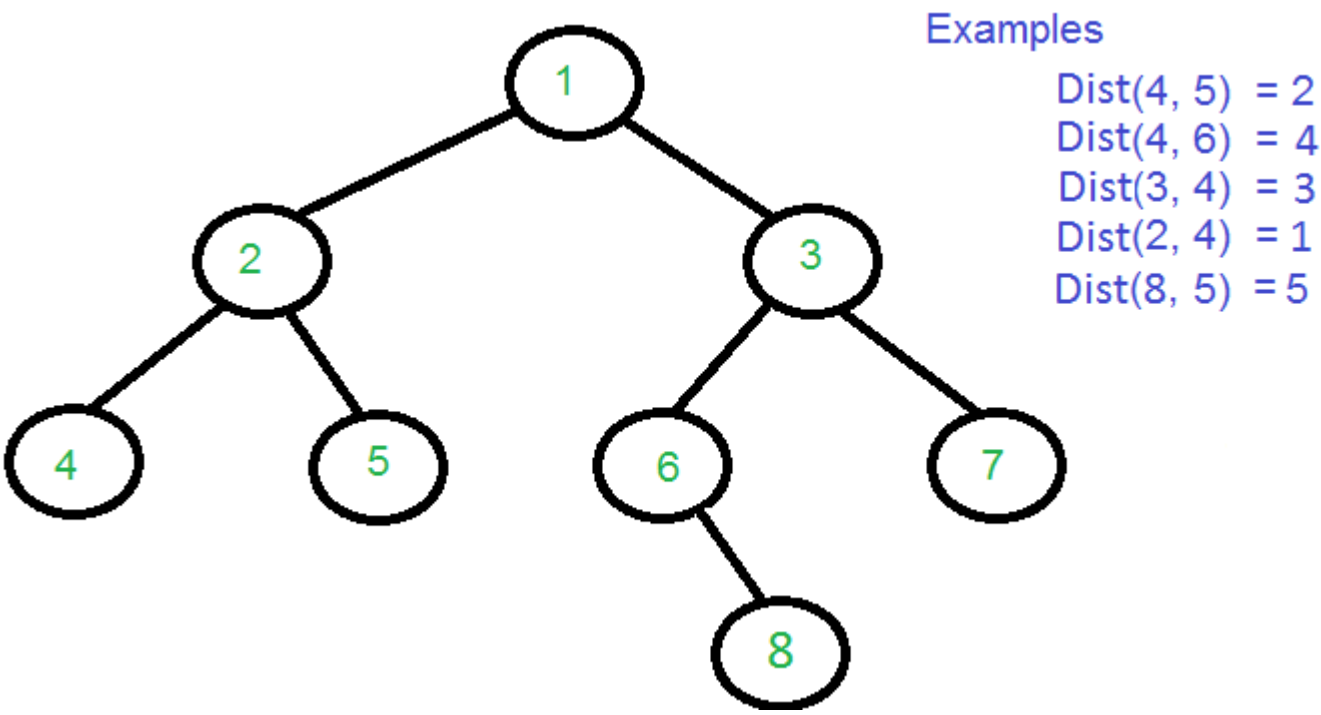
https://www.geeksforgeeks.org/queries-find-distance-two-nodes-binary-tree/

Chapter 259

Queries to find distance between two nodes of a Binary tree – O(logn) method

Queries to find distance between two nodes of a Binary tree - O(logn) method - GeeksforGeeks

Given a binary tree, the task is to find the distance between two keys in a binary tree, no parent pointers are given. Distance between two nodes is the minimum number of edges to be traversed to reach one node from other.
This problem has been already discussed in previous post but it uses **three traversals** of the Binary tree, one for finding Lowest Common Ancestor(LCA) of two nodes(let A and B) and then two traversals for finding distance between LCA and A and LCA and B which has O(n) time complexity. In this post, a method will be discussed that requires the **O(log(n))** time to find LCA of two nodes.



The distance between two nodes can be obtained in terms of lowest common ancestor. Following is the formula.

```
Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2*Dist(root, lca)
'n1' and 'n2' are the two given keys
'root' is root of given Binary Tree.
'lca' is lowest common ancestor of n1 and n2
Dist(n1, n2) is the distance between n1 and n2.
```

Above formula can also be written as:

```
Dist(n1, n2) = Level[n1] + Level[n2] - 2*Level[lca]
```

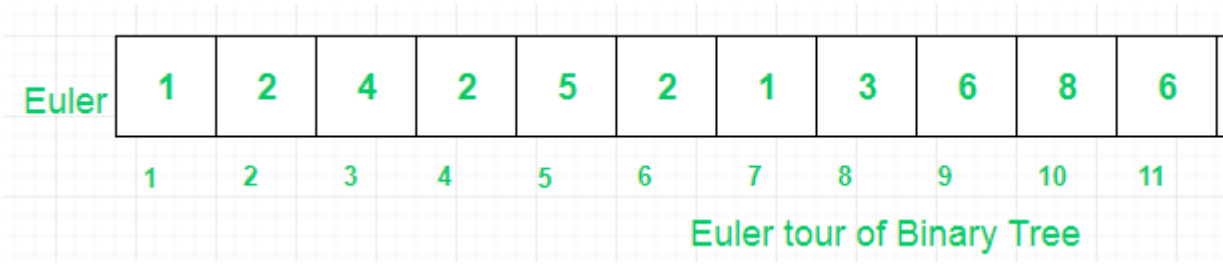
This problem can be breakdown into:

1. Finding levels of each node
2. Finding the Euler tour of binary tree
3. Building segment tree for LCA,

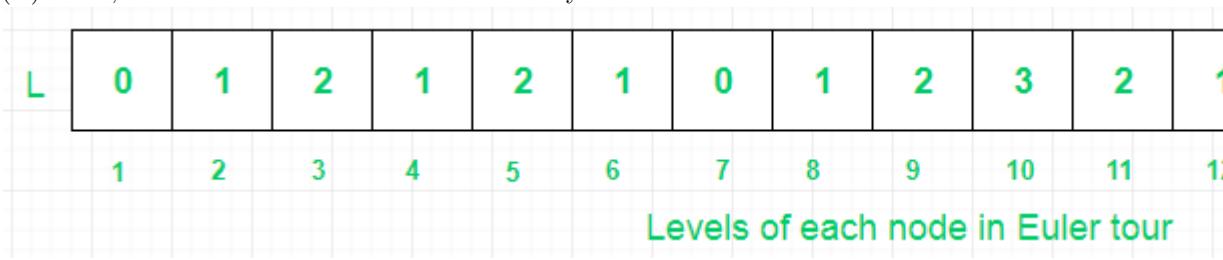
These steps are explained below :

1. Find the levels of each node by applying level order traversal.

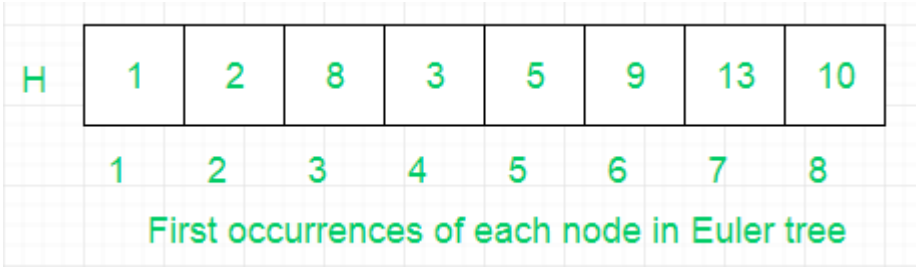
2. Find the LCA of two nodes in binary tree in O(logn) by Storing Euler tour of Binary tree in array and computing two other arrays with the help of levels of each node and Euler tour.
These steps are shown below:
(I) First, find Euler Tour of binary tree.



Euler tour of binary tree in example
(II) Then, store levels of each node in Euler array.



(III) Then, store First occurrences of all nodes of binary tree in Euler array. H stores the indices of nodes from Euler array, so that range of query for finding minimum can be minimized and their by further optimizing the query time.



3. Then ***build segment tree on L array*** and take the low and high values from H array that will give us the first occurrences of say Two nodes(A and B) . Then, *we query segment tree to find the minimum value* say X in range (**H[A] to H[B]**). Then **we use the index of value X as index to Euler array to get *LCA***, i.e. Euler[index(X)].
Let, A = 8 and B = 5.
(I) H[8] = 1 and H[5] =2
(II) Querying on Segment tree, we get min value in L array between 1 and 2 as X=0, index=7
(III) Then, LCA= Euler[7], i.e LCA = 1.

4. Finally, we apply distance formula discussed above to get distance between two nodes.

```
 // C++ program to find distance between
// two nodes for multiple queries
#include <bits/stdc++.h>
#define MAX 100001
using namespace std;

/* A tree node structure */
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

/* Utility function to create a new Binary Tree node */
struct Node* newNode(int data)
{
    struct Node* temp = new struct Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Array to store level of each node
int level[MAX];

// Utility Function to store level of all nodes
void FindLevels(struct Node* root)
{
    if (!root)
        return;

    // queue to hold tree node with level
    queue<pair<struct Node*, int> > q;

    // let root node be at level 0
    q.push({ root, 0 });

    pair<struct Node*, int> p;

    // Do level Order Traversal of tree
    while (!q.empty()) {
        p = q.front();
        q.pop();

        // Node p.first is on level p.second
```

```
        level[p.first->data] = p.second;

        // If left child exits, put it in queue
        // with current_level +1
        if (p.first->left)
            q.push({ p.first->left, p.second + 1 });

        // If right child exists, put it in queue
        // with current_level +1
        if (p.first->right)
            q.push({ p.first->right, p.second + 1 });
    }
}

// Stores Euler Tour
int Euler[MAX];

// index in Euler array
int idx = 0;

// Find Euler Tour
void eulerTree(struct Node* root)
{

    // store current node's data
    Euler[++idx] = root->data;

    // If left node exists
    if (root->left) {

        // traverse left subtree
        eulerTree(root->left);

        // store parent node's data
        Euler[++idx] = root->data;
    }

    // If right node exists
    if (root->right) {
        // traverse right subtree
        eulerTree(root->right);

        // store parent node's data
        Euler[++idx] = root->data;
    }
}

// checks for visited nodes
```

```
int vis[MAX];

// Stores level of Euler Tour
int L[MAX];

// Stores indices of first occurrence
// of nodes in Euler tour
int H[MAX];

// Preprocessing Euler Tour for finding LCA
void preprocessEuler(int size)
{
    for (int i = 1; i <= size; i++) {
        L[i] = level[Euler[i]];

        // If node is not visited before
        if (vis[Euler[i]] == 0) {
            // Add to first occurrence
            H[Euler[i]] = i;

            // Mark it visited
            vis[Euler[i]] = 1;
        }
    }
}

// Stores values and positions
pair<int, int> seg[4 * MAX];

// Utility function to find minimum of
// pair type values
pair<int, int> min(pair<int, int> a,
                   pair<int, int> b)
{
    if (a.first <= b.first)
        return a;
    else
        return b;
}

// Utility function to build segment tree
pair<int, int> buildSegTree(int low, int high, int pos)
{
    if (low == high) {
        seg[pos].first = L[low];
        seg[pos].second = low;
        return seg[pos];
    }
```

```
    int mid = low + (high - low) / 2;
    buildSegTree(low, mid, 2 * pos);
    buildSegTree(mid + 1, high, 2 * pos + 1);

    seg[pos] = min(seg[2 * pos], seg[2 * pos + 1]);
}

// Utility function to find LCA
pair<int, int> LCA(int qlow, int qhigh, int low,
                   int high, int pos)
{
    if (qlow <= low && qhigh >= high)
        return seg[pos];

    if (qlow > high || qhigh < low)
        return { INT_MAX, 0 };

    int mid = low + (high - low) / 2;

    return min(LCA(qlow, qhigh, low, mid, 2 * pos),
               LCA(qlow, qhigh, mid + 1, high, 2 * pos + 1));
}

// Function to return distance between
// two nodes n1 and n2
int findDistance(int n1, int n2, int size)
{
    // Maintain original Values
    int prevn1 = n1, prevn2 = n2;

    // Get First Occurrence of n1
    n1 = H[n1];

    // Get First Occurrence of n2
    n2 = H[n2];

    // Swap if low > high
    if (n2 < n1)
        swap(n1, n2);

    // Get position of minimum value
    int lca = LCA(n1, n2, 1, size, 1).second;

    // Extract value out of Euler tour
    lca = Euler[lca];

    // return calculated distance
    return level[prevn1] + level[prevn2] - 2 * level[lca];
```

```
}

void preProcessing(Node* root, int N)
{
    // Build Tree
    eulerTree(root);

    // Store Levels
    FindLevels(root);

    // Find L and H array
    preprocessEuler(2 * N - 1);

    // Build segment Tree
    buildSegTree(1, 2 * N - 1, 1);
}

/* Driver function to test above functions */
int main()
{
    int N = 8; // Number of nodes

    /* Constructing tree given in the above figure */
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);

    // Function to do all preprocessing
    preProcessing(root, N);

    cout << "Dist(4, 5) = " << findDistance(4, 5, 2 * N - 1) << "\n";
    cout << "Dist(4, 6) = " << findDistance(4, 6, 2 * N - 1) << "\n";
    cout << "Dist(3, 4) = " << findDistance(3, 4, 2 * N - 1) << "\n";
    cout << "Dist(2, 4) = " << findDistance(2, 4, 2 * N - 1) << "\n";
    cout << "Dist(8, 5) = " << findDistance(8, 5, 2 * N - 1) << "\n";

    return 0;
}
```

Output:

```
Dist(4, 5) = 2
```

```
Dist(4, 6) = 4
Dist(3, 4) = 3
Dist(2, 4) = 1
Dist(8, 5) = 5
```

Time Complexity: O(Log N)
Space Complexity: O(N)

Queries to find distance between two nodes of a Binary tree – O(1) method

Source

https://www.geeksforgeeks.org/queries-find-distance-two-nodes-binary-tree-ologn-method/

Chapter 260

Queries to insert, delete one occurrence of a number and print the least and most frequent element

Queries to insert, delete one occurrence of a number and print the least and most frequent element - GeeksforGeeks

Given Q queries of type 1, 2, 3 and 4 as described below.

- **Type-1:** Insert a number to the list.
- **Type-2:** Delete only one occurrence of a number if exists.
- **Type-3:** Print the least frequent element, if multiple elements exist then print the greatest among them.
- **Type-4:** Print the most frequent element, if multiple elements exist then print the smallest among them.

The task is to write a program to perform all the above queries.

Examples:

Input:
Query1: 1 6
Query2: 1 6
Query3: 1 7
Query4: 3
Query5: 1 7
Query6: 2 7
Query7: 1 7

Query8: 3
Query9: 4

Output:
7
7
6

While answering Query4, the frequency of 6 is 2 and that of
7 is 1, hence the least frequent element is 7.
In Query8, the least frequent element is 6 and 7, so print the largest.
In Query9, the most frequent element is 6 and 7, so print the smallest.

A **naive approach** is to use any Data-Structures(array, vector, ..) and store all the elements. Using a hash-table, the frequency of every element can be stored. While processing the Query of type-2, delete one occurrence of that element from the DS in which the element has been stored. The queries of type-3 and type-4 can be answered by traversing the hash-table. The time complexity will be **O(N)** per query, where N is the number of elements till then in the DS.

An **efficient approach** will be to use set container to answer every query. Using two sets, one hash-table the above problem can be solved in **O(log n)** per query. Two sets *s1* and *s2* are used, one stores the ***{num, frequency}***, while the other stores the ***{frequency, number}***. A hash-map is used which stores the frequency of each number. Design the set s2 using operator overloading such that it is sorted in ascending order of the first elements. If the first element appears to be same of one or more element, the set will be sorted in descending order of the second elements. The user-defined operating-overloading function thus will be:

```
bool operator b.second;
        return a.first < b.first;
}
Note: Operator overloading only works with user-defined
data-types. pr is a struct which has first and second as two integers.
```

Below is the algorithm to solve query of every type:

- **Type1:** Check using hash-table if the element exists. If it does not exist, then mark the number in hash-table. Insert **{num, 1}** in set s1 and **{1, num}** in set2 usinginsert(). If it exists previously, then get the frequency from hash-table and delete **{num, frequency}** from set1 and **{frequency, num}** from set2 using find() and erase() function. Insert **{num, frequency+1}** in set1 and **{frequency+1, num}** in set2. Also, increase the count in hash-table.
- **Type2:** Follow the same process as above in query type1. Ony difference is to decrease the count in hash-table and insert **{num, frequency-1}** in set1 and **{frequency-1, num}** in set2.
- **Type3:** Print the beginning element which can be obtained using begin() function, as the set has been designed in such a way that begin() returns the least frequent element. If there are more than one, then it returns the largest.

- **Type4:** Print the last element in the set which can be obtained using rbegin() function in set.

Below is the implementation of the above approach:

```
 // C++ program for performing
// Queries of insert, delete one
//  occurrence of a number and
// print the least and most frequent element
#include <bits/stdc++.h>
using namespace std;

// user-defined data-types
struct pr {
    int first;
    int second;
};

// user-defined function to
// design a set
bool operator<(pr a, pr b)
{
    if (a.first == b.first)
        return a.second > b.second;
    return a.first < b.first;
}

// declare a user-defined set
set<pr> s1, s2;

// hash map
unordered_map<int, int> m;

// Function to process the query
// of type-1
void type1(int num)
{

    // if the element is already there
    if (m[num]) {

        // get the frequency of the element
        int cnt = m[num];

        // returns an iterator pointing to
        // position where the pair is
        auto it1 = s1.find({ num, cnt });
        auto it2 = s2.find({ cnt, num });
```

```
        // deletes the pair from sets
        s1.erase(it1);
        s2.erase(it2);

        // re-insert the pair by increasing
        // frequency
        s1.insert({ num, m[num] + 1 });
        s2.insert({ m[num] + 1, num });
    }

    // if the element is not there in the list
    else {

        // insert the element with frequency 1
        s1.insert({ num, 1 });
        s2.insert({ 1, num });
    }

    // increase the count in hash-table
    m[num] += 1;
}

// Function to process the query
// of type-2
void type2(int num)
{
    // if the element exists
    if (m[num]) {

        // get the frequency of the element
        int cnt = m[num];

        // returns an iterator pointing to
        // position where the pair is
        auto it1 = s1.find({ num, cnt });
        auto it2 = s2.find({ cnt, num });

        // deletes the pair from sets
        s1.erase(it1);
        s2.erase(it2);

        // re-insert the pair by increasing
        // frequency
        s1.insert({ num, m[num] - 1 });
        s2.insert({ m[num] - 1, num });

        // decrease the count
```

```
        m[num] -= 1;
    }
}

// Function to process the query
// of type-3
int type3()
{
    // if the set is not empty
    // return the first element
    if (!s1.empty()) {
        auto it = s2.begin();
        return it->second;
    }

    else
        return -1;
}

// Function to process the query
// of type-4
int type4()
{

    // if the set is not empty
    // return the last element
    if (!s1.empty()) {
        auto it = s2.rbegin();
        return it->second;
    }

    else
        return -1;
}
// Driver Code
int main()
{

    // Queries

    // inserts 6, 6 and 7
    type1(6);
    type1(6);
    type1(7);

    // print the answer to query of type3
    cout << type3() << endl;
```

```
    // inserts 7
    type1(7);

    // deletes one occurence of 7
    type2(7);

    // inserts 7
    type1(7);

    // print the answer to query of type3
    cout << type3() << endl;

    // print the answer to query of type4
    cout << type4() << endl;

    return 0;
}
```

Output:

```
7
7
6
```

Time Complexity: O(log N) per query.
Auxiliary Space: O(N)

Source

https://www.geeksforgeeks.org/queries-to-insert-delete-one-occurrence-of-a-number-and-print-the-least-and-most-f

Chapter 261

Queue based approach for first non-repeating character in a stream

Queue based approach for first non-repeating character in a stream - GeeksforGeeks

Given a stream of characters and we have to find first non repeating character each time a character is inserted to the stream.

Examples:

```
Input  : a a b c
Output : a -1 b b

Input  : a a c
Output : a -1 c
```

We have already discussed a Doubly linked list based approach in the previous post.

Approach-

1. Create a count array of size 26(assuming only lower case characters are present) and initialize it with zero.
2. Create a queue of char datatype.
3. Store each character in queue and increase its frequency in the hash array.
4. For every character of stream, we check front of the queue.
5. If the frequency of character at the front of queue is one, then that will be the first non repeating character.
6. Else if frequency is more than 1, then we pop that element.
7. If queue became empty that means there are no non repeating character so we will print -1.

C++

```
 // C++ program for a Queue based approach to find first non-repeating
// character
#include <bits/stdc++.h>
using namespace std;
const int CHAR_MAX = 26;

// function to find first non repeating
// character of sa stream
void firstnonrepeating(char str[])
{
    queue<char> q;
    int charCount[CHAR_MAX] = { 0 };

    // traverse whole stream
    for (int i = 0; str[i]; i++) {

        // push each character in queue
        q.push(str[i]);

        // increment the frequency count
        charCount[str[i]-'a']++;

        // check for the non pepeating character
        while (!q.empty())
        {
            if (charCount[q.front()-'a'] > 1)
                q.pop();
            else
            {
                cout << q.front() << " ";
                break;
            }
        }

        if (q.empty())
            cout << -1 << " ";
    }
    cout << endl;
}

// Driver function
int main()
{
    char str[] = "aabc";
    firstnonrepeating(str);
    return 0;
```

```
}
```

Java

```
 //Java Program for a Queue based approach to find first non-repeating
//character

import java.util.LinkedList;
import java.util.Queue;

public class NonReapatingCQueue
{
     final static int CHAR_MAX = 26;

    // function to find first non repeating
    // character of stream
     static void firstNonRepeating(String str)
     {
         //count array of size 26(assuming only lower case characters are present)
         int[] charCount = new int[CHAR_MAX];

         //Queue to store Characters
         Queue<Character> q = new LinkedList<Character>();

        // traverse whole stream
         for(int i=0; i<str.length(); i++)
         {
             char c = str.charAt(i);

            // push each character in queue
             q.add(c);

            // increment the frequency count
             charCount++;

            // check for the non repeating character
             while(!q.isEmpty())
             {
                 if(charCount[q.peek() - 'a'] > 1)
                     q.remove();
                 else
                 {
                     System.out.print(q.peek() + " ");
                     break;
                 }
             }
             if(q.isEmpty())
                 System.out.print(-1 + " ");
```

```
            }
            System.out.println();
        }

    // Driver function
    public static void main(String[] args)
    {
        String str = "aabc";
        firstNonRepeating(str);
    }

}
//This code is Contributed by Sumit Ghosh
```

Output:

```
a -1 b b
```

Source

https://www.geeksforgeeks.org/queue-based-approach-for-first-non-repeating-character-in-a-stream/

Chapter 262

Range Queries for Frequencies of array elements

Range Queries for Frequencies of array elements - GeeksforGeeks

Given an array of n non-negative integers. The task is to find frequency of a particular element in the arbitrary range of array[]. The range is given as positions (not 0 based indexes) in array. There can be multiple queries of given type. For example:-

```
Input  : arr[] = {2, 8, 6, 9, 8, 6, 8, 2, 11};
         left = 2, right = 8, element = 8
         left = 2, right = 5, element = 6
Output : 3
         1
The element 8 appears 3 times in arr[left-1..right-1]
The element 6 appears 1 time in arr[left-1..right-1]
```

Naive approach: is to traverse from left to right and update count variable whenever we find the element.

Below is the code of Naive approach:-

C++

```
 // C++ program to find total count of an element
// in a range
#include<bits/stdc++.h>
using namespace std;

// Returns count of element in arr[left-1..right-1]
int findFrequency(int arr[], int n, int left,
```

```
                          int right, int element)
{
    int count = 0;
    for (int i=left-1; i<=right; ++i)
        if (arr[i] == element)
            ++count;
    return count;
}

// Driver Code
int main()
{
    int arr[] = {2, 8, 6, 9, 8, 6, 8, 2, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Print frequency of 2 from position 1 to 6
    cout << "Frequency of 2 from 1 to 6 = "
         << findFrequency(arr, n, 1, 6, 2) << endl;

    // Print frequency of 8 from position 4 to 9
    cout << "Frequency of 8 from 4 to 9 = "
         << findFrequency(arr, n, 4, 9, 8);

    return 0;
}
```

Java

```
 // JAVA Code to find total count of an element
// in a range

class GFG {

    // Returns count of element in arr[left-1..right-1]
    public static int findFrequency(int arr[], int n,
                                int left, int right,
                                      int element)
    {
        int count = 0;
        for (int i = left - 1; i < right; ++i)
            if (arr[i] == element)
                ++count;
        return count;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
```

```
        int arr[] = {2, 8, 6, 9, 8, 6, 8, 2, 11};
        int n = arr.length;

        // Print frequency of 2 from position 1 to 6
        System.out.println("Frequency of 2 from 1 to 6 = " +
             findFrequency(arr, n, 1, 6, 2));

        // Print frequency of 8 from position 4 to 9
        System.out.println("Frequency of 8 from 4 to 9 = " +
             findFrequency(arr, n, 4, 9, 8));

    }
  }
// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python program to find total
# count of an element in a range

# Returns count of element
# in arr[left-1..right-1]
def findFrequency(arr, n, left, right, element):

    count = 0
    for i in range(left - 1, right):
        if (arr[i] == element):
            count += 1
    return count


# Driver Code
arr = [2, 8, 6, 9, 8, 6, 8, 2, 11]
n = len(arr)

# Print frequency of 2 from position 1 to 6
print("Frequency of 2 from 1 to 6 = ",
        findFrequency(arr, n, 1, 6, 2))

# Print frequency of 8 from position 4 to 9
print("Frequency of 8 from 4 to 9 = ",
        findFrequency(arr, n, 4, 9, 8))


# This code is contributed by Anant Agarwal.
```

C#

```
 // C# Code to find total count
// of an element in a range
using System;

class GFG {

    // Returns count of element
    // in arr[left-1..right-1]
    public static int findFrequency(int []arr, int n,
                                    int left, int right,
                                    int element)
    {
        int count = 0;
        for (int i = left - 1; i < right; ++i)
            if (arr[i] == element)
                ++count;
        return count;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {2, 8, 6, 9, 8, 6, 8, 2, 11};
        int n = arr.Length;

        // Print frequency of 2
        // from position 1 to 6
        Console.WriteLine("Frequency of 2 from 1 to 6 = " +
                          findFrequency(arr, n, 1, 6, 2));

        // Print frequency of 8
        // from position 4 to 9
        Console.Write("Frequency of 8 from 4 to 9 = " +
                      findFrequency(arr, n, 4, 9, 8));

    }
}

// This code is contributed by Nitin Mittal.
```

Output:

```
 Frequency of 2 from 1 to 6 = 1
 Frequency of 8 from 4 to 9 = 2
```

Time complexity of this approach is O(right – left + 1) or O(n)
Auxiliary space: O(1)

An **Efficient approach** is to use hashing. In C++, we can use unordered_map

1. At first, we will store the position in map[] of every distinct element as a vector like that

```
int arr[] = {2, 8, 6, 9, 8, 6, 8, 2, 11};
map[2] = {1, 8}
map[8] = {2, 5, 7}
map[6] = {3, 6}
ans so on...
```

2. As we can see that elements in map[] are already in sorted order (Because we inserted elements from left to right), the answer boils down to find the total count in that hash map[] using binary search like method.
3. In C++ we can use lower_bound which will returns an iterator pointing to the first element in the range [first, last] which has a value not less than 'left'. and upper_bound returns an iterator pointing to the first element in the range [first,last) which has a value greater than 'right'.
4. After that we just need to subtract the upper_bound() and lower_bound() result to get the final answer. For example, suppose if we want to find the total count of 8 in the range from [1 to 6], then the map[8] of lower_bound() function will return the result 0 (pointing to 2) and upper_bound() will return 2 (pointing to 7), so we need to subtract the both the result like 2 – 0 = 2 .

Below is C++ code of above approach

```
 // C++ program to find total count of an element
#include<bits/stdc++.h>
using namespace std;

unordered_map< int, vector<int> > store;

// Returns frequency of element in arr[left-1..right-1]
int findFrequency(int arr[], int n, int left,
                      int right, int element)
{
    // Find the position of first occurrence of element
    int a = lower_bound(store[element].begin(),
                        store[element].end(),
                        left)
            - store[element].begin();

    // Find the position of last occurrence of element
    int b = upper_bound(store[element].begin(),
                        store[element].end(),
```

```
                        right)
            - store[element].begin();

    return b-a;
}

// Driver code
int main()
{
    int arr[] = {2, 8, 6, 9, 8, 6, 8, 2, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Storing the indexes of an element in the map
    for (int i=0; i<n; ++i)
        store[arr[i]].push_back(i+1); //starting index from 1

    // Print frequency of 2 from position 1 to 6
    cout << "Frequency of 2 from 1 to 6 = "
         << findFrequency(arr, n, 1, 6, 2) <<endl;

    // Print frequency of 8 from position 4 to 9
    cout << "Frequency of 8 from 4 to 9 = "
         << findFrequency(arr, n, 4, 9, 8);

    return 0;
}
```

Output:

```
Frequency of 2 from 1 to 6 = 1
Frequency of 8 from 4 to 9 = 2
```

This approach will be beneficial if we have a large number of queries of an arbitrary range asking the total frequency of particular element.

Time complexity: O(log N) for single query.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/range-queries-for-frequencies-of-array-elements/

Chapter 263

Rearrange an array such that arr[i] = i

Rearrange an array such that arr[i] = i - GeeksforGeeks

Given an array of elements of length N, ranging from 1 to N. All elements may not be present in the array. If element is not present then there will be -1 present in the array. Rearrange the array such that A[i] = i and if i is not present, display -1 at that place.

Examples:

```
Input : arr = {-1, -1, 6, 1, 9, 3, 2, -1, 4, -1}
Output : [-1, 1, 2, 3, 4, -1, 6, -1, -1, 9]

Input : arr = {19, 7, 0, 3, 18, 15, 12, 6, 1, 8,
               11, 10, 9, 5, 13, 16, 2, 14, 17, 4}
Output : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
         11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Approach:
1. Navigate the array.
2. Check if a[i] = -1, if yes then ignore it.
3. If a[i] != -1, Check if element a[i] is at its **correct position (i=A[i])**. If yes then ignore it.
4. If a[i] != -1 and element a[i] is not at its correct **position (i!=A[i])** then place it to its correct position, but there are two conditions:

- Either **A[i] is vacate**, means **A[i] = -1**, then just put **A[i] = i** .
- OR **A[i] is not vacate**, means **A[i] = x**, then **int y=x** put **A[i] = i**. Now, we need to place **y** to its correct place, so repeat from step 3.

Below is the implementation for above approach:

C++

```
 // CPP program for rearrange an
// array such that arr[i] = i.
#include <bits/stdc++.h>
using namespace std;

// Function to rearrange an array
// such that arr[i] = i.
int fix(int A[], int len)
{
    for (int i = 0; i < len; i++)
    {
        if (A[i] != -1 && A[i] != i)
        {
            int x = A[i];

            // check if desired place
            // is not vacate
            while (A[x] != -1 && A[x] != x)
            {
                // store the value from
                // desired place
                int y = A[x];

                // place the x to its correct
                // position
                A[x] = x;

                // now y will become x, now
                // search the place for x
                x = y;
            }

            // place the x to its correct
            // position
            A[x] = x;

            // check if while loop hasn't
            // set the correct value at A[i]
            if (A[i] != i)
            {

                // if not then put -1 at
                // the vacated place
                A[i] = -1;
```

```
            }
        }
    }
}

// Driver function.
int main()
{
    int A[] = { -1, -1, 6, 1, 9,
                3, 2, -1, 4, -1 };

    int len = sizeof(A) / sizeof(A[0]);
    fix(A, len);

    for (int i = 0; i < len; i++)
        cout << A[i] << " ";
}

// This code is contributed by Smitha Dinesh Semwal
```

Java

```
 // Java program for rearrange an
// array such that arr[i] = i.
import java.util.*;
import java.lang.*;

public class GfG {

    // Function to rearrange an array
    // such that arr[i] = i.
    public static int[] fix(int[] A)
    {
        for (int i = 0; i < A.length; i++)
        {
            if (A[i] != -1 && A[i] != i)
            {
                int x = A[i];

                // check if desired place
                // is not vacate
                while (A[x] != -1 && A[x] != x)
                {

                    // store the value from
                    // desired place
                    int y = A[x];
```

```
                    // place the x to its correct
                    // position
                    A[x] = x;

                    // now y will become x, now
                    // search the place for x
                    x = y;
                }

                // place the x to its correct
                // position
                A[x] = x;

                // check if while loop hasn't
                // set the correct value at A[i]
                if (A[i] != i)
                {

                    // if not then put -1 at
                    // the vacated place
                    A[i] = -1;
                }
            }
        }
        return A;
    }

    // Driver function.
    public static void main(String[] args)
    {
        int A[] = {-1, -1, 6, 1,
                    9, 3, 2, -1, 4,-1};
        System.out.println(Arrays.toString(fix(A)));
    }
}
```

Python3

```
 # Python3 program for rearrange an
# array such that arr[i] = i.

# Function to rearrange an array
# such that arr[i] = i.
def fix( A, len):

    for i in range(0, len):

        if (A[i] != -1 and A[i] != i):
```

```
            x = A[i];

            # check if desired place
            # is not vacate
            while (A[x] != -1 and A[x] != x):
                #store the value from
                # desired place
                y = A[x]

                # place the x to its correct
                # position
                A[x] = x

                # now y will become x, now
                # search the place for x
                x = y

            # place the x to its correct
            # position
            A[x] = x;

            # check if while loop hasn't
            # set the correct value at A[i]
            if (A[i] != i) :

                # if not then put -1 at
                # the vacated place
                A[i] = -1;

# Driver function.
A = [ -1, -1, 6, 1, 9, 3, 2, -1, 4, -1 ]

fix(A, len(A))

for i in range(0, len(A)):
    print (A[i],end = ' ')

# This code is contributed by Saloni1297
```

C#

```
 // C# program for rearrange
// an array such that
// arr[i] = i.
using System;

class GfG
{
```

```
// Function to rearrange an
// array such that arr[i] = i.
public static int[] fix(int[] A)
{
    for (int i = 0;
             i < A.Length; i++)
    {
        if (A[i] != -1 &&
            A[i] != i)
        {
            int x = A[i];

            // check if desired
            // place is not vacate
            while (A[x] != -1 &&
                   A[x] != x)
            {

                // store the value
                // from desired place
                int y = A[x];

                // place the x to its
                // correct position
                A[x] = x;

                // now y will become x,
                // now search the place
                // for x
                x = y;
            }

            // place the x to its
            // correct position
            A[x] = x;

            // check if while loop
            // hasn't set the correct
            // value at A[i]
            if (A[i] != i)
            {

                // if not then put -1
                // at the vacated place
                A[i] = -1;
            }
        }
    }
```

```
    return A;
}

// Driver Code
static void Main()
{
    int []A = new int[]{-1, -1, 6, 1, 9,
                        3, 2, -1, 4,-1};
    Console.WriteLine(string.Join(",",
                                  fix(A)));
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP program for rearrange an
// array such that arr[i] = i.

// Function to rearrange an
// array such that arr[i] = i.
function fix(&$A, $len)
{
    for ($i = 0; $i < $len; $i++)
    {
        if ($A[$i] != -1 &&
            $A[$i] != $i)
        {
            $x = $A[$i];

            // check if desired
            // place is not vacate
            while ($A[$x] != -1 &&
                   $A[$x] != $x)
            {
                // store the value
                // from desired place
                $y = $A[$x];

                // place the x to its
                // correct position
                $A[$x] = $x;

                // now y will become x,
                // now search the place
```

```
                // for x
                $x = $y;
            }

            // place the x to its
            // correct position
            $A[$x] = $x;

            // check if while loop hasn't
            // set the correct value at A[i]
            if ($A[$i] != $i)
            {
                // if not then put -1
                // at the vacated place
                $A[$i] = -1;
            }
        }
    }
}

// Driver Code
$A = array(-1, -1, 6, 1, 9,
            3, 2, -1, 4, -1);

$len = count($A);
fix($A, $len);

for ($i = 0; $i < $len; $i++)
    echo ($A[$i]." ");

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
[-1, 1, 2, 3, 4, -1, 6, -1, -1, 9]
```

Another Approach (Using HashSet) :
1) Store all the numbers present in the array into a HashSet
2) Iterate through the length of the array, if the corresponding position element is present in the HashSet, then set A[i] = i, else A[i] = -1

Below is the implementation of above approach.

Java

```
 // Java program for rearrange an
// array such that arr[i] = i.
import java.util.*;
import java.lang.*;

class GfG {

    // Function to rearrange an array
    // such that arr[i] = i.
    public static int[] fix(int[] A)
    {
        Set<Integer> s = new HashSet<Integer>();

        // Storing all the values in the HashSet
        for(int i = 0; i < A.length; i++)
        {
           s.add(A[i]);
        }

        for(int i = 0; i < A.length; i++)
        {
           if(s.contains(i))
             A[i] = i;
           else
             A[i] = -1;
        }

      return A;
}

    // Driver function.
    public static void main(String[] args)
    {
        int A[] = {-1, -1, 6, 1, 9,
                    3, 2, -1, 4,-1};

        // Function calling
        System.out.println(Arrays.toString(fix(A)));
    }
}
```

Python3

```
 # Python3program for rearrange an
# array such that arr[i] = i.


# Function to rearrange an array
```

```
# such that arr[i] = i.
def fix(A):
    s = set()

    # Storing all the values in the Set
    for i in range(len(A)):
        s.add(A[i])

    for i in range(len(A)):
        # check for item if present in set
        if i in s:
            A[i] = i
        else:
            A[i] = -1
    return A

# Driver code
if __name__ == "__main__":
    A = [-1, -1, 6, 1, 9,
        3, 2, -1, 4,-1]
    print(fix(A))
```

C#

```
 // C# program for rearrange an
// array such that arr[i] = i.
using System;

class GfG
{

    // Function to rearrange
    // an array such that
    // arr[i] = i.
    static int[] fix(int []A)
    {
        for (int i = 0;
                  i < A.Length; i++)
        {
            if (A[i] != -1 &&
                A[i] != i)
            {
                int x = A[i];

                // check if desired place
                // is not vacate
                while (A[x] != -1 &&
                       A[x] != x)
```

```
                {

                    // store the value from
                    // desired place
                    int y = A[x];

                    // place the x to its
                    // correct position
                    A[x] = x;

                    // now y will become x, now
                    // search the place for x
                    x = y;
                }

                // place the x to its
                // correct position
                A[x] = x;

                // check if while loop hasn't
                // set the correct value at A[i]
                if (A[i] != i)
                {
                    // if not then put -1 at
                    // the vacated place
                    A[i] = -1;
                }
            }
        }
        return A;
    }

    // Driver Code
    static void Main()
    {
        int []A = new int[]{-1, -1, 6, 1, 9,
                            3, 2, -1, 4,-1};
        Console.Write(string.Join(",", fix(A)));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
[-1, 1, 2, 3, 4, -1, 6, -1, -1, 9]
```

Improved By : manishshaw1, rahulpy, Venkata

Source

https://www.geeksforgeeks.org/rearrange-array-arri/

Chapter 264

Recaman's sequence

Recaman's sequence - GeeksforGeeks

Given an integer n. Print first n elements of Recaman's sequence.

Examples:

```
Input : n = 6
Output : 0, 1, 3, 6, 2, 7

Input  : n = 17
Output : 0, 1, 3, 6, 2, 7, 13, 20, 12, 21,
         11, 22, 10, 23, 9, 24, 8
```

It is basically a function with domain and co-domain as natural numbers and 0. It is recursively defined as below:
Specifically, let a(n) denote the (n+1)-th term. (0 being already there).
The rule says:

```
a(0) = 0,
if n > 0 and the number is not
   already included in the sequence,
     a(n) = a(n - 1) - n
else
     a(n) = a(n-1) + n.
```

Below is simple implementation where we store all n Recaman Sequence numbers in an array. We compute next number using recursive formula mentioned above.

C++

```
 // C++ program to print n-th number in Recaman's
// sequence
#include <bits/stdc++.h>
using namespace std;

// Prints first n terms of Recaman sequence
int recaman(int n)
{
    // Create an array to store terms
    int arr[n];

    // First term of the sequence is always 0
    arr[0] = 0;
    printf("%d, ", arr[0]);

    // Fill remaining terms using recursive
    // formula.
    for (int i=1; i< n; i++)
    {
        int curr = arr[i-1] - i;
        int j;
        for (j = 0; j < i; j++)
        {
            // If arr[i-1] - i is negative or
            // already exists.
            if ((arr[j] == curr) || curr < 0)
            {
                curr = arr[i-1] + i;
                break;
            }
        }

        arr[i] = curr;
        printf("%d, ", arr[i]);
    }
}

// Driver code
int main()
{
    int n = 17;
    recaman(n);
    return 0;
}
```

Java

```
 // Java program to print n-th number in Recaman's
```

```
// sequence
import java.io.*;

class GFG {

    // Prints first n terms of Recaman sequence
    static void recaman(int n)
    {
        // Create an array to store terms
        int arr[] = new int[n];

        // First term of the sequence is always 0
        arr[0] = 0;
        System.out.print(arr[0]+" ,");

        // Fill remaining terms using recursive
        // formula.
        for (int i = 1; i < n; i++)
        {
            int curr = arr[i - 1] - i;
            int j;
            for (j = 0; j < i; j++)
            {
                // If arr[i-1] - i is negative or
                // already exists.
                if ((arr[j] == curr) || curr < 0)
                {
                    curr = arr[i - 1] + i;
                    break;
                }
            }

            arr[i] = curr;
            System.out.print(arr[i]+", ");

        }
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 17;
        recaman(n);

    }
}

// This code is contributed by vt_m
```

Python 3

```
 # Python 3 program to print n-th
# number in Recaman's sequence

# Prints first n terms of Recaman
# sequence
def recaman(n):

    # Create an array to store terms
    arr = [0] * n

    # First term of the sequence
    # is always 0
    arr[0] = 0
    print(arr[0], end=", ")

    # Fill remaining terms using
    # recursive formula.
    for i in range(1, n):

        curr = arr[i-1] - i
        for j in range(0, i):

            # If arr[i-1] - i is
            # negative or already
            # exists.
            if ((arr[j] == curr) or curr < 0):
                curr = arr[i-1] + i
                break

        arr[i] = curr
        print(arr[i], end=", ")

# Driver code
n = 17

recaman(n)

# This code is contributed by Smitha.
```

C#

```
 // C# program to print n-th number in Recaman's
// sequence
using System;
```

```
class GFG {

    // Prints first n terms of Recaman sequence
    static void recaman(int n)
    {
        // Create an array to store terms
        int []arr = new int[n];

        // First term of the sequence is always 0
        arr[0] = 0;
        Console.Write(arr[0]+" ,");

        // Fill remaining terms using recursive
        // formula.
        for (int i = 1; i < n; i++)
        {
            int curr = arr[i - 1] - i;
            int j;
            for (j = 0; j < i; j++)
            {
                // If arr[i-1] - i is negative or
                // already exists.
                if ((arr[j] == curr) || curr < 0)
                {
                    curr = arr[i - 1] + i;
                    break;
                }
            }

            arr[i] = curr;
        Console.Write(arr[i]+", ");

        }
    }

    // Driver code
    public static void Main ()
    {
        int n = 17;
        recaman(n);

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to print n-th
// number in Recaman's sequence

// Prints first n terms
// of Recaman sequence
function recaman($n)
{

    // First term of the
    // sequence is always 0
    $arr[0] = 0;
    echo $arr[0], ", ";

    // Fill remaining terms
    // using recursive formula.
    for ($i = 1; $i < $n; $i++)
    {
            $curr = $arr[$i - 1] - $i;
            $j;
        for ($j = 0; $j < $i; $j++)
        {

            // If arr[i-1] - i
            // is negative or
            // already exists.
            if (($arr[$j] == $curr) || $curr < 0)
            {
                $curr = $arr[$i-1] + $i;
                break;
            }
        }

        $arr[$i] = $curr;
        echo $arr[$i], ", ";
    }
}

    // Driver Code
    $n = 17;
    recaman($n);

// This code is contributed by Ajit
?>
```

Output:

```
0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8,
```

Time Complexity : $O(n^2)$
Auxiliary Space : O(n)

Optimizations :
We can us hashing to store previously computed values and can make this program work in O(n) time.

```
 // C++ program to print n-th number in Recaman's
// sequence
#include <bits/stdc++.h>
using namespace std;

// Prints first n terms of Recaman sequence
void recaman(int n)
{
    if (n <= 0)
      return;

    // Print first term and store it in a hash
    printf("%d, ", 0);
    unordered_set<int> s;
    s.insert(0);

    // Print remaining terms using recursive
    // formula.
    int prev = 0;
    for (int i=1; i< n; i++)
    {
        int curr = prev - i;

        // If arr[i-1] - i is negative or
        // already exists.
        if (curr < 0 || s.find(curr) != s.end())
           curr = prev + i;

        s.insert(curr);

        printf("%d, ", curr);
        prev = curr;
    }
}

// Driver code
int main()
{
    int n = 17;
    recaman(n);
```

```
    return 0;
}
```

Output:

```
0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8,
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/recamans-sequence/

Chapter 265

Remove minimum number of elements such that no common element exist in both array

Remove minimum number of elements such that no common element exist in both array - GeeksforGeeks

Given two arrays A[] and B[] consisting of n and m elements respectively. Find minimum number of elements to remove from each array such that no common element exist in both.

Examples:

```
Input : A[] = { 1, 2, 3, 4}
        B[] = { 2, 3, 4, 5, 8 }
Output : 3
We need to remove 2, 3 and 4 from any array.

Input : A[] = { 4, 2, 4, 4}
        B[] = { 4, 3 }
Output : 1
We need to remove 4 from from B[]

Input : A[] = { 1, 2, 3, 4 }
        B[] = { 5, 6, 7 }
Output : 0
There is no common element in both.
```

Count occurrence of each number in both array. If there is number in both array remove number from array in which it appear less number of times add it to the result.

C++

```
 // CPP program to find minimum element
// to remove so no common element
// exist in both array
#include <bits/stdc++.h>
using namespace std;

// To find no elements to remove
// so no common element exist
int minRemove(int a[], int b[], int n, int m)
{
    // To store count of array element
    unordered_map<int, int> countA, countB;

    // Count elements of a
    for (int i = 0; i < n; i++)
        countA[a[i]]++;

    // Count elements of b
    for (int i = 0; i < m; i++)
        countB[b[i]]++;

    // Traverse through all common element, and
    // pick minimum occurrence from two arrays
    int res = 0;
    for (auto x : countA)
        if (countB.find(x.first) != countB.end())
            res += min(x.second, countB[x.first]);

    // To return count of minimum elements
    return res;
}

// Driver program to test minRemove()
int main()
{
    int a[] = { 1, 2, 3, 4 };
    int b[] = { 2, 3, 4, 5, 8 };
    int n = sizeof(a) / sizeof(a[0]);
    int m = sizeof(b) / sizeof(b[0]);

    cout << minRemove(a, b, n, m);

    return 0;
}
```

Java

```
 // JAVA Code to Remove minimum number of elements
```

```
// such that no common element exist in both array
import java.util.*;

class GFG {

    // To find no elements to remove
    // so no common element exist
    public static int minRemove(int a[], int b[], int n,
                                                int m)
    {
        // To store count of array element
        HashMap<Integer, Integer> countA = new HashMap<
                                        Integer, Integer>();
        HashMap<Integer, Integer> countB = new HashMap<
                                        Integer, Integer>();

        // Count elements of a
        for (int i = 0; i < n; i++){
           if (countA.containsKey(a[i]))
                countA.put(a[i], countA.get(a[i]) + 1);

           else countA.put(a[i], 1);

        }

        // Count elements of b
        for (int i = 0; i < m; i++){
             if (countB.containsKey(b[i]))
                    countB.put(b[i], countB.get(b[i]) + 1);

               else countB.put(b[i], 1);
        }

        // Traverse through all common element, and
        // pick minimum occurrence from two arrays
        int res = 0;

        Set<Integer> s = countA.keySet();

        for (int x : s)
            if(countB.containsKey(x))
                res += Math.min(countB.get(x),
                                countA.get(x));

        // To return count of minimum elements
        return res;
    }
```

```
    /* Driver program to test above function */
    public static void main(String[] args)
    {

            int a[] = { 1, 2, 3, 4 };
            int b[] = { 2, 3, 4, 5, 8 };
            int n = a.length;
            int m = b.length;

            System.out.println(minRemove(a, b, n, m));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/remove-minimum-number-elements-no-common-element-exist-array/

Chapter 266

Reorder the given string to form a K-concatenated string

Reorder the given string to form a K-concatenated string - GeeksforGeeks

Given a string S and an integer K. The task is to form a string T such that the string T is a reordering of the string S in a way that it is a **K-Concatenated-String**. A string is said to be a K-Concatenated-String if it contains exactly K copies of some string.

For example, the string "geekgeek" is a 2-Concatenated-String formed by concatenating 2 copies of the string "geek".

Note: Multiple answers are possible.

Examples:

> **Input** : s = "gkeekgee", k=2
> **Output**: geekgeek
> eegkeegk is another possible K-Concatenated-String
>
> **Input**: s = "abcd", k=2
>
> **Output**: Not Possible

Approach: To find a valid ordering that is a K-Concatenated-string, iterate through the entire string and maintain a frequency array for the characters to hold the number of times each character occurs in a string. Since, in a K-Concatenated-string, the number of times a character occurs should be divisible by K. If any character is found not following this, then the string cant be ordered in any way to represent a K-Concatenated-string, else there can be exactly **(Frequency of i^{th} character / K)** copies of the i^{th} character in a single copy of the k-Concatenated-string. Such single copies when repeated K times form a valid K-Concatenated-string.

Below is the implementation of the above approach:

C++

```
 // C++ program to form a
// K-Concatenated-String from a given string
#include <bits/stdc++.h>
using namespace std;

// Function to print the k-concatenated string
string kStringGenerate(string str, int k)
{

    // maintain a frequency table
    // for lowercase letters
    int frequency[26] = { 0 };

    int len = str.length();

    for (int i = 0; i < len; i++) {

        // for each character increment its frequency
        // in the frequency array
        frequency[str[i] - 'a']++;
    }

    // stores a single copy of a string,
    // which on repeating forms a k-string
    string single_copy = "";

    // iterate over frequency array
    for (int i = 0; i < 26; i++) {

        // if the character occurs in the string,
        // check if it is divisible by k,
        // if not divisible then k-string cant be formed
        if (frequency[i] != 0) {

            if ((frequency[i] % k) != 0) {

                string ans = "Not Possible";
                return ans;
            }
            else {

                // ith character occurs (frequency[i]/k) times in a
                // single copy of k-string
                int total_occurrences = (frequency[i] / k);

                for (int j = 0; j < total_occurrences; j++) {
                    single_copy += char(i + 97);
                }
```

```
            }
        }
    }

    string kString;

    // append the single copy formed k times
    for (int i = 0; i < k; i++) {
        kString += single_copy;
    }

    return kString;
}

// Driver Code
int main()
{
    string str = "gkeekgee";
    int K = 2;

    string kString = kStringGenerate(str, K);
    cout << kString;
    return 0;
}
```

Java

```
 // Java program to form a
// K-Concatenated-String
// from a given string
import java.io.*;
import java.util.*;
import java.lang.*;

class GFG
{

// Function to print
// the k-concatenated string
static String kStringGenerate(String str,
                              int k)
{

    // maintain a frequency table
    // for lowercase letters
    int frequency[] = new int[26];
    Arrays.fill(frequency,0);
    int len = str.length();
```

```
for (int i = 0; i < len; i++)
{

    // for each character
    // increment its frequency
    // in the frequency array
    frequency[str.charAt(i) - 'a']++;
}

// stores a single copy
// of a string, which on
// repeating forms a k-string
String single_copy = "";

// iterate over
// frequency array
for (int i = 0; i < 26; i++)
{

    // if the character occurs
    // in the string, check if
    // it is divisible by k,
    // if not divisible then
    // k-string cant be formed
    if (frequency[i] != 0)
    {

        if ((frequency[i] % k) != 0)
        {
            String ans = "Not Possible";
            return ans;
        }
        else
        {

            // ith character occurs
            // (frequency[i]/k) times in
            // a single copy of k-string
            int total_occurrences = (frequency[i] / k);

            for (int j = 0;
                     j < total_occurrences; j++)
            {
                single_copy += (char)(i + 97);
            }
        }
    }
```

```
    }

    String kString = "";

    // append the single
    // copy formed k times
    for (int i = 0; i < k; i++)
    {
        kString += single_copy;
    }

    return kString;
}

// Driver Code
public static void main(String[] args)
{
    String str = "gkeekgee";
    int K = 2;

    String kString = kStringGenerate(str, K);
    System.out.print(kString);
}
}
```

Output:

```
eegkeegk
```

Time Complexity: O(N), where N is the length of the string.

Source

https://www.geeksforgeeks.org/reorder-the-given-string-to-form-a-k-concatenated-string/

Chapter 267

Return maximum occurring character in an input string

Return maximum occurring character in an input string - GeeksforGeeks

Write an efficient function to return maximum occurring character in the input string e.g., if input string is “test” then function should return ‘t’.

Algorithm:
One obvious approach to solve this problem would be to sort the input string and then traverse through the sorted string to find the character which is occurring maximum number of times. Let us see if we can improve on this. So we will use a technique called ‘Hashing’. In this, when we traverse through the string, we would hash each character into an array of ASCII characters.

```
Input string = "test"
1: Construct character count array from the input string.
  count['e'] = 1
  count['s'] = 1
  count['t'] = 2

2: Return the index of maximum value in count array (returns ‘t’).
```

Typically, ASCII characters are 256, so we use our Hash array size as 256. But if we know that our input string will have characters with value from 0 to 127 only, we can limit Hash array size as 128. Similarly, based on extra info known about input string, the Hash array size can be limited to 26.

Implementation:

C++

```
// C++ program to output the maximum occurring character
```

```
// in a string
#include<bits/stdc++.h>
#define ASCII_SIZE 256
using namespace std;

char getMaxOccuringChar(char* str)
{
    // Create array to keep the count of individual
    // characters and initialize the array as 0
    int count[ASCII_SIZE] = {0};

    // Construct character count array from the input
    // string.
    int len = strlen(str);
    for (int i=0; i<len; i++)
        count[str[i]]++;

    int max = -1;  // Initialize max count
    char result;   // Initialize result

    // Traversing through the string and maintaining
    // the count of each character
    for (int i = 0; i < len; i++) {
        if (max < count[str[i]]) {
            max = count[str[i]];
            result = str[i];
        }
    }

    return result;
}

// Driver program to test the above function
int main()
{
    char str[] = "sample string";
    cout << "Max occurring character is "
         << getMaxOccuringChar(str);
}
```

Java

```
 // Java program to output the maximum occurring character
// in a string

public class GFG
{
    static final int ASCII_SIZE = 256;
```

```
    static char getMaxOccuringChar(String str)
    {
        // Create array to keep the count of individual
        // characters and initialize the array as 0
        int count[] = new int[ASCII_SIZE];

        // Construct character count array from the input
        // string.
        int len = str.length();
        for (int i=0; i<len; i++)
            count[str.charAt(i)]++;

        int max = -1;  // Initialize max count
        char result = ' ';   // Initialize result

        // Traversing through the string and maintaining
        // the count of each character
        for (int i = 0; i < len; i++) {
            if (max < count[str.charAt(i)]) {
                max = count[str.charAt(i)];
                result = str.charAt(i);
            }
        }

        return result;
    }

    // Driver Method
    public static void main(String[] args)
    {
        String str = "sample string";
        System.out.println("Max occurring character is " +
                           getMaxOccuringChar(str));
    }
}
```

Python

```
 # Python program to return the maximum occuring character in the input string
ASCII_SIZE = 256

def getMaxOccuringChar(str):
    # Create array to keep the count of individual characters
    # Initialize the count array to zero
    count = [0] * ASCII_SIZE

    # Utility variables
    max = -1
```

```
    c = ''

    # Traversing through the string and maintaining the count of
    # each character
    for i in str:
        count[ord(i)]+=1;

    for i in str:
        if max < count[ord(i)]:
            max = count[ord(i)]
            c = i

    return c

# Driver program to test the above function
str = "sample string"
print "Max occurring character is " + getMaxOccuringChar(str)

# Although this program can be written in atmost 3 lines in Python
# the above program has been written for a better understanding of
# the reader

# Shorter version of the program
# import collections
# str = "sample string"
# print "Max occurring character is " +
#        collections.Counter(str).most_common(1)[0][0]

# This code has been contributed by Bhavya Jain
```

C#

```
 // C# program to output the maximum
// occurring character in a string
using System;

class GFG
{
    static int ASCII_SIZE = 256;

    static char getMaxOccuringChar(String str)
    {
        // Create array to keep the count of
        // individual characters and
        // initialize the array as 0
        int []count = new int[ASCII_SIZE];

        // Construct character count array
```

```
        // from the input string.
        int len = str.Length;
        for (int i = 0; i < len; i++)
            count[str[i]]++;

        int max = -1; // Initialize max count
        char result = ' '; // Initialize result

        // Traversing through the string and
        // maintaining the count of each character
        for (int i = 0; i < len; i++) {
            if (max < count[str[i]]) {
                max = count[str[i]];
                result = str[i];
            }
        }

        return result;
    }

    // Driver Method
    public static void Main()
    {
        String str = "sample string";
        Console.Write("Max occurring character is " +
                            getMaxOccuringChar(str));
    }
}

// This code is contributed by Sam007
```

Output:

```
Max occurring character is s
```

Time Complexity: O(n)
Space Complexity: O(1) — Because we are using fixed space (Hash array) irrespective of input string size.

Notes:
If more than one characters have the same count and that count is maximum then the function returns the first character with maximum count in input string. For example if input string is “test sample” then there are three characters with same and maximum count two i.e. “t”, “e” and “s” but our program will result “t” because “t” comes first in input string. Similarly, the output for “cbbbbccc” would be “c”.

As an variation to the above program, think about the change in code if you want to output “e” for input “test sample” i.e. the character of maximum count and that character should be of least ASCII value. For “cbbbbccc”, output should be “b”. Try it out !

Also, can you think of improvement if we can avoid two loopings in the above? Basically, you need to figure out if we can solve the same problem with one loop itself instead of two loops.

Source

https://www.geeksforgeeks.org/return-maximum-occurring-character-in-the-input-string/

Chapter 268

Root to leaf path with maximum distinct nodes

Root to leaf path with maximum distinct nodes - GeeksforGeeks

Given a Binary Tree, find count of distinct nodes in a root to leaf path with maximum distinct nodes.
Examples:

```
Input :   1
        /    \
       2      3
      / \    / \
     4   5  6   3
              \   \
               8   9
Output : 4
The root to leaf path with maximum distinct
nodes is 1-3-6-8.
```

A **simple solution** is to explore all root to leaf paths. In every root to leaf path, count distinct nodes and finally return the maximum count.

An **efficient solution** is to use hashing. We recursively traverse the tree and maintain count of distinct nodes on path from root to current node. We recur for left and right subtrees and finally return maximum of two values.

Below c++ implementation of above idea

```
 // C++ program to find count of distinct nodes
// on a path with maximum distinct nodes.
#include <bits/stdc++.h>
```

```
using namespace std;

// A node of binary tree
struct Node {
    int data;
    struct Node *left, *right;
};

// A utility function to create a new Binary
// Tree node
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

int largestUinquePathUtil(Node* node, unordered_map<int, int> m)
{
    if (!node)
        return m.size();

    // put this node into hash
    m[node->data]++;

    int max_path = max(largestUinquePathUtil(node->left, m),
                       largestUinquePathUtil(node->right, m));

    // remove current node from path "hash"
    m[node->data]--;

    // if we reached a condition where all duplicate value
    // of current node is deleted
    if (m[node->data] == 0)
        m.erase(node->data);

    return max_path;
}

// A utility function to find long unique value path
int largestUinquePath(Node* node)
{
    if (!node)
        return 0;

    // hash that store all node value
    unordered_map<int, int> hash;
```

```
    // return max length unique value path
    return largestUinquePathUtil(node, hash);
}

// Driver program to test above functions
int main()
{
    // Create binary tree shown in above figure
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);

    cout << largestUinquePath(root) << endl;

    return 0;
}
```

```
4
```

Time Complexity :O(n)

Source

https://www.geeksforgeeks.org/root-leaf-path-maximum-distinct-nodes/

Chapter 269

Root to leaf paths having equal lengths in a Binary Tree

Root to leaf paths having equal lengths in a Binary Tree - GeeksforGeeks

Given a binary tree, print the number of root to leaf paths having equal lengths.

Examples:

```
Input : Root of below tree
                    10
                   /   \
                  8     2
                 /  \   /  \
                3    5 2   4
Output : 4 paths are of length 3.

Input : Root of below tree
                    10
                   /   \
                  8     2
                 /  \   /  \
                3    5 2   4
               /             \
              9               1
Output : 2 paths are of length 3
         2 paths are of length 4
```

The idea is to traverse the tree and keep track of path length. Whenever we reach a leaf node, we increment path length count in a hash map.
Once we have traverse the tree, hash map has counts of distinct path lengths. Finally we print contents of hash map.

```
 // C++ program to count root to leaf paths of different
// lengths.
#include<bits/stdc++.h>
using namespace std;

/* A binary tree node */
struct Node
{
    int data;
    struct Node* left, *right;
};

/* utility that allocates a new node with the
   given data and NULL left and right pointers. */
struct Node* newnode(int data)
{
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right  = NULL;
    return (node);
}

// Function to store counts of different root to leaf
// path lengths in hash map m.
void pathCountUtil(Node *node, unordered_map<int, int> &m,
                                            int path_len)
{
    // Base condition
    if (node == NULL)
        return;

    // If leaf node reached, increment count of path
    // length of this root to leaf path.
    if (node->left == NULL && node->right == NULL)
    {
         m[path_len]++;
         return;
    }

    // Recursively call for left and right subtrees with
    // path lengths more than 1.
    pathCountUtil(node->left, m, path_len+1);
    pathCountUtil(node->right, m, path_len+1);
}

// A wrapper over pathCountUtil()
void pathCounts(Node *root)
{
```

```
   // create an empty hash table
   unordered_map<int, int> m;

   // Recursively check in left and right subtrees.
   pathCountUtil(root, m, 1);

   // Print all path lenghts and their counts.
   for (auto itr=m.begin(); itr != m.end(); itr++)
      cout << itr->second << " paths have length "
           << itr->first << endl;
}

// Driver program to run the case
int main()
{
    struct Node *root = newnode(8);
    root->left    = newnode(5);
    root->right   = newnode(4);
    root->left->left = newnode(9);
    root->left->right = newnode(7);
    root->right->right = newnode(11);
    root->right->right->left = newnode(3);
    pathCounts(root);
    return 0;
}
```

Output:

```
4 paths have length 3
```

Source

https://www.geeksforgeeks.org/root-leaf-paths-equal-lengths-binary-tree/

Chapter 270

Second most repeated word in a sequence

Second most repeated word in a sequence - GeeksforGeeks

Given a sequence of strings, the task is to find out the second most repeated (or frequent) string in the given sequence.(Considering no two words are the second most repeated, there will be always a single word).

Examples:

```
Input : {"aaa", "bbb", "ccc", "bbb",
         "aaa", "aaa"}
Output : bbb

Input : {"geeks", "for", "geeks", "for",
          "geeks", "aaa"}
Output : for
```

Asked in : Amazon

1. Store all the words in a map with their occurrence with word as key and its occurrence as value.
2. Find the second largest value in the map.
3. Traverse the map again and return the word with occurrence value equals to second max value.

C++

```
 // C++ program to find out the second
// most repeated word
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the word
string secMostRepeated(vector<string> seq)
{

    // Store all the words with its occurrence
    unordered_map<string, int> occ;
    for (int i = 0; i < seq.size(); i++)
        occ[seq[i]]++;

    // find the second largest occurrence
    int first_max = INT_MIN, sec_max = INT_MIN;
    for (auto it = occ.begin(); it != occ.end(); it++) {
        if (it->second > first_max) {
            sec_max = first_max;
            first_max = it->second;
        }

        else if (it->second > sec_max &&
                 it->second != first_max)
            sec_max = it->second;
    }

    // Return string with occurrence equals
    // to sec_max
    for (auto it = occ.begin(); it != occ.end(); it++)
        if (it->second == sec_max)
            return it->first;
}

// Driver program
int main()
{
    vector<string> seq = { "ccc", "aaa", "ccc",
                           "ddd", "aaa", "aaa" };
    cout << secMostRepeated(seq);
    return 0;
}
```

Java

```
 // Java program to find out the second
// most repeated word

import java.util.*;
```

```
class GFG
{
    // Method to find the word
    static String secMostRepeated(Vector<String> seq)
    {
        // Store all the words with its occurrence
        HashMap<String, Integer> occ = new HashMap<String,Integer>(seq.size()){
            @Override
            public Integer get(Object key) {
                 return containsKey(key) ? super.get(key) : 0;
            }
        };

        for (int i = 0; i < seq.size(); i++)
            occ.put(seq.get(i), occ.get(seq.get(i))+1);

        // find the second largest occurrence
       int first_max = Integer.MIN_VALUE, sec_max = Integer.MIN_VALUE;

       Iterator<Map.Entry<String, Integer>> itr = occ.entrySet().iterator();
       while (itr.hasNext())
       {
           Map.Entry<String, Integer> entry = itr.next();
           int v = entry.getValue();
           if( v > first_max) {
                sec_max = first_max;
                first_max = v;
            }

            else if (v > sec_max &&
                     v != first_max)
                sec_max = v;
       }

       // Return string with occurrence equals
        // to sec_max
       itr = occ.entrySet().iterator();
       while (itr.hasNext())
       {
           Map.Entry<String, Integer> entry = itr.next();
           int v = entry.getValue();
           if (v == sec_max)
                return entry.getKey();
       }

       return null;
    }
```

```
    // Driver method
    public static void main(String[] args)
    {
        String arr[] = { "ccc", "aaa", "ccc",
                         "ddd", "aaa", "aaa" };
        List<String> seq =  Arrays.asList(arr);

        System.out.println(secMostRepeated(new Vector<>(seq)));
    }
}
// This program is contributed by Gaurav Miglani
```

Output:

```
ccc
```

Reference:
https://www.careercup.com/question?id=5748104113422336

Source

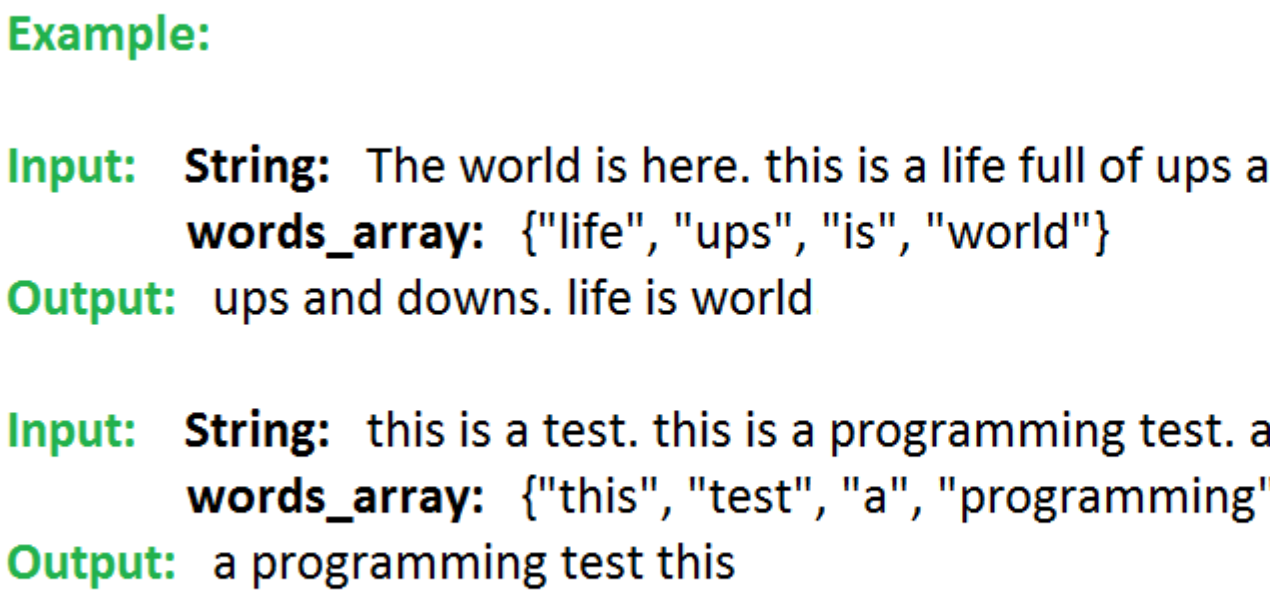
https://www.geeksforgeeks.org/second-repeated-word-sequence/

Chapter 271

Shortest substring of a string containing all given words

Shortest substring of a string containing all given words - GeeksforGeeks

Print the shortest sub-string of a string containing all the given words.



In the first example, two solutions are possible: “world is here. this is a life full of ups” and “ups and downs. life is world”.

1. Initialize HashMap with all the given words which are required to be searched and assign their values as -1.
2. Maintain a counter.
3. Traverse the entire String word by word and do following for each sentence word
    - If the sentence word exists in the list of words you’re looking for, update the last position of that word.
    - Increase the total count if the updated last position was not initialized.
    - If the total count is equal to count of all given words, loop through the last positions and find the smallest one. The distance between the current position and that value will be the length of the substring. Record these values and find the best one over all positions

Below is Java implementation of above steps.

```
 // Java program to find the shortest substring of a
// string containing all given words using HashMap
import java.io.*;
import java.util.*;
import java.util.HashMap;

class Shortest {

    public static void findShortest(String sentence,
                                    String[] words)
    {
        // Make an array of words from given sentence
        // We remove punctuations before splitting.
        String replicate = sentence.replace(".", "");
        replicate = replicate.replace(", ", "");
        replicate = replicate.replace("!", "");
        String sent_words[] = replicate.split(" ");

        // hashmap to store given words in a map.
        HashMap<String, Integer> map = new HashMap<>();
        int length = words.length;
        for (int i = 0; i < length; i++)
            map.put(words[i], -1);

        // Traverse through all sentence words
        // and if they match with given words
        // then mark their appearances in map.
        int len_sub = Integer.MAX_VALUE;
        int count = 0;
        int local_start = 0, local_end = 0;
```

```
    for (int i = 0; i < sent_words.length; i++) {
        if (map.containsKey(sent_words[i]) == true) {

            // If this is the first occurrence
            int index = map.get(sent_words[i]);
            if (index == -1)
                count++;

            // Store latest index
            map.put(sent_words[i], i);

            // If all words matched
            if (count == length) {

                // Find smallest index
                int min = Integer.MAX_VALUE;
                for (Map.Entry<String, Integer> m :
                                map.entrySet()) {
                    int val = m.getValue();
                    if (val < min)
                        min = val;
                }

                // Check if current length is smaller
                // then length so far
                int s = i - min;
                if (s < len_sub) {
                    local_start = min;
                    local_end = i;
                }
            }
        }
    }

    // Printing original substring (with punctuations)
    // using resultant local_start and local_end.
    String[] original_parts = sentence.split(" ");
    for (int i = local_start; i < local_end; i++)
        System.out.print(original_parts[i] + " ");
}

// Driver code
public static void main(String args[])
{
    String sentence = "The world is here. this is a" +
        " life full of ups and downs. life is world.";
    String[] words = { "life", "ups", "is", "world" };
    findShortest(sentence, words);
```

```
    }
}
```

Output :

```
ups and downs. life is
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/shortest-substring-string-containing-given-words/

Chapter 272

Smallest element in an array that is repeated exactly 'k' times.

Smallest element in an array that is repeated exactly 'k' times. - GeeksforGeeks

Given an array of size n, the goal is to find out the smallest number that is repeated exactly 'k' times where k > 0?
Assume that array has only positive integers and 1 <= arr[i] < 1000 for each i = 0 to n -1.

Examples:

```
Input : arr[] = {2 2 1 3 1}
        k = 2
Output: 1
Explanation:
Here in array,
2 is repeated 2 times
1 is repeated 2 times
3 is repeated 1 time
Hence 2 and 1 both are repeated 'k' times
i.e 2 and min(2, 1) is 1

Input : arr[] = {3 5 3 2}
        k = 1
Output : 2
Explanation:
Both 2 and 5 are repeating 1 time but
min(5, 2) is 2
```

Simple Approach: A simple approach is to use two nested loops.The outer loop picks an

element one by one starting from the leftmost element. The inner loop checks if the same element is present on right side of it. If present increase the count and make the number negative which we got at the right side to prevent it from counting again.

C++

```
 // C++ program to find smallest number
// in array that is repeated exactly
// 'k' times.
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

int findDuplicate(int arr[], int n, int k)
{
    // Since arr[] has numbers in range from
    // 1 to MAX
    int res = MAX + 1;
    for (int i = 0; i < n; i++) {
        if (arr[i] > 0) {

            // set count to 1 as number is present
            // once
            int count = 1;
            for (int j = i + 1; j < n; j++)
                if (arr[i] == arr[j])
                    count += 1;

            // If frequency of number is equal to 'k'
            if (count == k)
                res = min(res, arr[i]);
        }
    }
    return res;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << findDuplicate(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to find smallest number
// in array that is repeated exactly
// 'k' times.
public class GFG {
    static final int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int arr[], int n, int k)
    {
        // Since arr[] has numbers in range from
        // 1 to MAX
        int res = MAX + 1;
        for (int i = 0; i < n; i++) {
            if (arr[i] > 0) {

                // set count to 1 as number is
                // present once
                int count = 1;
                for (int j = i + 1; j < n; j++)
                    if (arr[i] == arr[j])
                        count += 1;

                // If frequency of number is equal
                // to 'k'
                if (count == k)
                    res = Math.min(res, arr[i]);
            }
        }
        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.length;
        System.out.println(findDuplicate(arr, n, k));
    }
}
// This article is contributed by Sumit Ghosh
```

Python3

```
 # Python 3 program to find smallest
# number in array that is repeated
# exactly 'k' times.
```

```
MAX = 1000

def findDuplicate(arr, n, k):

    # Since arr[] has numbers in
    # range from 1 to MAX
    res = MAX + 1

    for i in range(0, n):
        if (arr[i] > 0):

            # set count to 1 as number
            # is present once
            count = 1
            for j in range(i + 1, n):
                if (arr[i] == arr[j]):
                    count += 1

            # If frequency of number is equal to 'k'
            if (count == k):
                res = min(res, arr[i])

    return res

# Driver code
arr = [2, 2, 1, 3, 1]
k = 2
n = len(arr)
print(findDuplicate(arr, n, k))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find smallest number
// in array that is repeated exactly
// 'k' times.
using System;

public class GFG {

    static int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int[] arr,
                             int n, int k)
    {
```

```
        // Since arr[] has numbers in range
        // from 1 to MAX
        int res = MAX + 1;

        for (int i = 0; i < n; i++)
        {
            if (arr[i] > 0)
            {

                // set count to 1 as number
                // is present once
                int count = 1;
                for (int j = i + 1; j < n; j++)
                    if (arr[i] == arr[j])
                        count += 1;

                // If frequency of number is
                // equal to 'k'
                if (count == k)
                    res = Math.Min(res, arr[i]);
            }
        }

        return res;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.Length;

        Console.WriteLine(
                      findDuplicate(arr, n, k));
    }
}

// This article is contributed by vt_m.
```

Output:

```
1
```

Time Complexity : $O(n^2)$
Auxiliary Space : O(1)
This solution doesn't require array elements to be in limited range.

Better Solution : Sort the input array and find the first element with exactly k count of appearances.

C++

```
 // C++ program to find smallest number
// in array that is repeated exactly
// 'k' times.
#include <bits/stdc++.h>
using namespace std;

int findDuplicate(int arr[], int n, int k)
{
    // Sort the array
    sort(arr, arr + n);

    // Find the first element with exactly
    // k occurrences.
    int i = 0;
    while (i < n) {
        int j, count = 1;
        for (j = i + 1; j < n && arr[j] == arr[i]; j++)
            count++;

        if (count == k)
            return arr[i];

        i = j;
    }

    return -1;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << findDuplicate(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to find smallest number
// in array that is repeated exactly
// 'k' times.
import java.util.Arrays;
public class GFG {

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int arr[], int n, int k)
    {
        // Sort the array
        Arrays.sort(arr);

        // Find the first element with exactly
        // k occurrences.
        int i = 0;
        while (i < n) {
            int j, count = 1;
            for (j = i + 1; j < n && arr[j] == arr[i]; j++)
                count++;

            if (count == k)
                return arr[i];

            i = j;
        }

        return -1;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.length;
        System.out.println(findDuplicate(arr, n, k));
    }
}
// This article is contributed by Sumit Ghosh
```

Python

```
 # Python program to find smallest number
# in array that is repeated exactly
# 'k' times.

def findDuplicate(arr, n, k):
```

```
    # Sort the array
    arr.sort()

    # Find the first element with exactly
    # k occurrences.
    i = 0
    while (i < n):
        j, count = i + 1, 1
        while (j < n and arr[j] == arr[i]):
            count += 1
            j += 1

        if (count == k):
            return arr[i]

        i = j

    return -1

# Driver code
arr = [ 2, 2, 1, 3, 1 ];
k = 2
n = len(arr)
print findDuplicate(arr, n, k)

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to find smallest number
// in array that is repeated exactly
// 'k' times.
using System;

public class GFG {

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int[] arr,
                             int n, int k)
    {

        // Sort the array
        Array.Sort(arr);

        // Find the first element with
        // exactly k occurrences.
```

```
        int i = 0;
        while (i < n) {
            int j, count = 1;
            for (j = i + 1; j < n &&
                     arr[j] == arr[i]; j++)
                count++;

            if (count == k)
                return arr[i];

            i = j;
        }

        return -1;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.Length;

        Console.WriteLine(
               findDuplicate(arr, n, k));
    }
}

// This article is contributed by vt_m.
```

Output:

```
1
```

Time Complexity : O(n Log n)
Auxiliary Space : O(1)

Efficient Approach : Efficient approach is based on the fact that array has numbers in small range (1 to 1000). We solve this problem by using a frequency array of size max and store the frequency of every number in that array.

C++

```
 // C++ program to find smallest number
// in array that is repeated exactly
// 'k' times.
```

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

int findDuplicate(int arr[], int n, int k)
{
    // Computing frequencies of all elements
    int freq[MAX];
    memset(freq, 0, sizeof(freq));
    for (int i = 0; i < n; i++) {
        if (arr[i] < 1 && arr[i] > MAX) {
            cout << "Out of range";
            return -1;
        }
        freq[arr[i]] += 1;
    }

    // Finding the smallest element with
    // frequency as k
    for (int i = 0; i < MAX; i++) {

        // If frequency of any of the number
        // is equal to k starting from 0
        // then return the number
        if (freq[i] == k)
            return i;
    }

    return -1;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << findDuplicate(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to find smallest number
// in array that is repeated exactly
// 'k' times.
public class GFG {
```

```
    static final int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int arr[], int n, int k)
    {
        // Computing frequencies of all elements
        int[] freq = new int[MAX];

        for (int i = 0; i < n; i++) {
            if (arr[i] < 1 && arr[i] > MAX) {
                System.out.println("Out of range");
                return -1;
            }
            freq[arr[i]] += 1;
        }

        // Finding the smallest element with
        // frequency as k
        for (int i = 0; i < MAX; i++) {

            // If frequency of any of the number
            // is equal to k starting from 0
            // then return the number
            if (freq[i] == k)
                return i;
        }

        return -1;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.length;
        System.out.println(findDuplicate(arr, n, k));
    }
}
// This article is contributed by Sumit Ghosh
```

Python

```
 # Python program to find smallest number
# in array that is repeated exactly
# 'k' times.
```

```
MAX = 1000

def findDuplicate(arr, n, k):

    # Computing frequencies of all elements
    freq = [0 for i in range(MAX)]

    for i in range(n):
        if (arr[i] < 1 and arr[i] > MAX):
            print "Out of range"
            return -1
        freq[arr[i]] += 1

    # Finding the smallest element with
    # frequency as k
    for i in range(MAX):

        # If frequency of any of the number
        # is equal to k starting from 0
        # then return the number
        if (freq[i] == k):
            return i

    return -1

# Driver code
arr = [ 2, 2, 1, 3, 1 ]
k = 2
n = len(arr)
print findDuplicate(arr, n, k)

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to find smallest number
// in array that is repeated exactly
// 'k' times.
using System;

public class GFG {

    static int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int[] arr,
```

```
                            int n, int k)
    {

        // Computing frequencies of all
        // elements
        int[] freq = new int[MAX];

        for (int i = 0; i < n; i++)
        {
            if (arr[i] < 1 && arr[i] > MAX)
            {
                Console.WriteLine("Out of range");
                return -1;
            }

            freq[arr[i]] += 1;
        }

        // Finding the smallest element with
        // frequency as k
        for (int i = 0; i < MAX; i++) {

            // If frequency of any of the
            // number is equal to k starting
            // from 0 then return the number
            if (freq[i] == k)
                return i;
        }

        return -1;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.Length;

        Console.WriteLine(
               findDuplicate(arr, n, k));
    }
}

// This article is contributed by vt_m.
```

Output:

```
1
```

Time Complexity: O(MAX + n)
Auxiliary Space : O(MAX)

Can we solve it in O(n) time if range is not limited?
Please see Smallest element repeated exactly 'k' times (not limited to small range)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/smallest-element-array-repeated-exactly-k-times/

Chapter 273

Smallest element repeated exactly ‘k’ times (not limited to small range)

Smallest element repeated exactly ‘k’ times (not limited to small range) - GeeksforGeeks

Given an array of size n, the goal is to find out the smallest number that is repeated exactly ‘k’ times where k > 0?
And

Examples:

```
Input : a[] = {2, 1, 3, 1, 2, 2}
        k = 3
Output : 2

Input : a[] = {3, 4, 3, 2, 1, 5, 5}
        k = 2
Output : 3
Explanation: As 3 is smaller than 5.
So 3 should be printed.
```

We have discussed different solutions of this problem in below post.

Smallest element in an array that is repeated exactly ‘k’ times

The solutions discussed above are either limited to small range work in more than linear time. In this post a hashing based solution is discussed that works in O(n) time and is applicable to any range. Below are abstract steps.

1) Create a hash map that stores elements and their frequencies.
2) Traverse given array. For every element being traversed, increment its frequency.

3) Traverse hash map and print the smallest element with frequency k.

C++

```
 // C++ program to find the smallest element
// with frequency exactly k.
#include <bits/stdc++.h>
using namespace std;

int smallestKFreq(int a[], int n, int k)
{
    unordered_map<int, int> m;

    // Map is used to store the count of
    // elements present in the array
    for (int i = 0; i < n; i++)
        m[a[i]]++;

    // Traverse the map and find minimum
    // element with frequency k.
    int res = INT_MAX;
    for (auto it = m.begin(); it != m.end(); ++it)
        if (it->second == k)
           res = min(res, it->first);

    return (res != INT_MAX)? res : -1;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << smallestKFreq(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to find the smallest element
// with frequency exactly k.
import java.util.*;

class GFG {

    public static int smallestKFreq(int a[], int n, int k)
```

```
    {
        HashMap<Integer, Integer> m = new HashMap<Integer, Integer>();

        // Map is used to store the count of
        // elements present in the array
        for (int i = 0; i < n; i ++)

            if (m.containsKey(a[i]))
                m.put(a[i], m.get(a[i]) + 1);

           else m.put(a[i], 1);

        // Traverse the map and find minimum
        // element with frequency k.
        int res = Integer.MAX_VALUE;
        Set<Integer> s = m.keySet();

        for (int temp : s)
            if (m.get(temp) == k)
               res = Math.min(res, temp);

        return (res != Integer.MAX_VALUE)? res : -1;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = { 2, 2, 1, 3, 1 };
        int k = 2;

        System.out.println(smallestKFreq(arr, arr.length, k));

    }
  }
// This code is contributed by Arnav Kr. Mandal.
```

Output:

```
1
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Related Article:
Smallest number repeating k times

Source

https://www.geeksforgeeks.org/smallest-element-repeated-exactly-k-times-not-limited-small-range/

Chapter 274

Smallest subarray with all occurrences of a most frequent element

Smallest subarray with all occurrences of a most frequent element - GeeksforGeeks

Given an array, A.Let x be an element in the array.x has the maximum frequency in the array.Find the smallest subsegment of the array which also has x as the maximum frequency element.

Examples:

```
Input :  arr[] = {4, 1, 1, 2, 2, 1, 3, 3}
Output :   1, 1, 2, 2, 1
The most frequent element is 1. The smallest
subarray that has all occurrences of it is
1 1 2 2 1

Input :  A[] = {1, 2, 2, 3, 1}
Output : 2, 2
Note that there are two elements that appear
two times, 1 and 2. The smallest window for
1 is whole array and smallest window for 2 is
{2, 2}. Since window for 2 is smaller, this is
our output.
```

Approach: Observe that if X is the maximum repeated element of our subsegment then the sunsegment should look like this [X, ..., X], cause if the subsegment end or begins with another element we can delete it which does not alter our answer.
To solve this problem, let us store for every distinct element in the array three values, index

of the first occurrence of the element and the index of the last occurrence the element and the frequency of the element. And at every step for a maximum repeated element minimize the size of our subsegment.

C++

```
 // C++ implementation to find smallest
// subarray with all occurrences of
// a most frequent element
#include <bits/stdc++.h>
using namespace std;

void smallestSubsegment(int a[], int n)
{
    // To store left most occurrence of elements
    unordered_map<int, int> left;

    // To store counts of elements
    unordered_map<int, int> count;

    // To store maximum frequency
    int mx = 0;

    // To store length and starting index of
    // smallest result window
    int mn, strindex;

    for (int i = 0; i < n; i++) {

        int x = a[i];

        // First occurrence of an element,
        // store the index
        if (count[x] == 0) {
            left[x] = i;
            count[x] = 1;
        }

        // increase the frequency of elements
        else
            count[x]++;

        // Find maximum repeated element and
        // store its last occurrence and first
        // occurrence
        if (count[x] > mx) {
            mx = count[x];
            mn = i - left[x] + 1; // length of subsegment
```

```
            strindex = left[x];
        }

        // select subsegment of smallest size
        else if (count[x] == mx && i - left[x] + 1 < mn) {
            mn = i - left[x] + 1;
            strindex = left[x];
        }
    }

    // Print the subsegment with all occurrences of
    // a most frequent element
    for (int i = strindex; i < strindex + mn; i++)
        cout << a[i] << " ";
}

// Driver code
int main()
{
    int A[] = { 1, 2, 2, 2, 1 };
    int n = sizeof(A) / sizeof(A[0]);
    smallestSubsegment(A, n);
    return 0;
}
```

Java

```
 // Java implementation to find smallest
// subarray with all occurrences of
// a most frequent element
import java.io.*;
import java.util.*;
class GfG {

    static void smallestSubsegment(int a[], int n)
    {
        // To store left most occurrence of elements
        HashMap<Integer, Integer> left= new HashMap<Integer, Integer>();

        // To store counts of elements
        HashMap<Integer, Integer> count= new HashMap<Integer, Integer>();

        // To store maximum frequency
        int mx = 0;

        // To store length and starting index of
        // smallest result window
        int mn = -1, strindex = -1;
```

```
    for (int i = 0; i < n; i++)
    {

        int x = a[i];

        // First occurrence of an element,
        // store the index
        if (count.get(x) == null)
        {
            left.put(x, i) ;
            count.put(x, 1);
        }

        // increase the frequency of elements
        else
            count.put(x, count.get(x) + 1);

        // Find maximum repeated element and
        // store its last occurrence and first
        // occurrence
        if (count.get(x) > mx)
        {
            mx = count.get(x);

            // length of subsegment
            mn = i - left.get(x) + 1;
            strindex = left.get(x);
        }

        // select subsegment of smallest size
        else if ((count.get(x) == mx) &&
                (i - left.get(x) + 1 < mn))
        {
            mn = i - left.get(x) + 1;
            strindex = left.get(x);
        }
    }

    // Print the subsegment with all occurrences of
    // a most frequent element
    for (int i = strindex; i < strindex + mn; i++)
        System.out.print(a[i] + " ");
}

// Driver program
public static void main (String[] args)
{
```

```
        int A[] = { 1, 2, 2, 2, 1 };
        int n = A.length;
        smallestSubsegment(A, n);
    }
}

// This code is contributed by Gitanjali.
```

Output:

```
2 2 2
```

Time Complexity : O(n)

Source

https://www.geeksforgeeks.org/smallest-subarray-with-all-occurrences-of-a-most-frequent-element/

Chapter 275

Smallest subarray with k distinct numbers

Smallest subarray with k distinct numbers - GeeksforGeeks

We are given an array a consisting of n integers and an integer k. We need to find minimum range in array [l, r] (both l and r are inclusive) such that there are exactly k different numbers.

Examples:

```
Input : arr[] = { 1, 1, 2, 2, 3, 3, 4, 5}
           k = 3
Output : 5 7

Input : arr[] = { 1, 2, 2, 3}
           k = 2
Output : 0 1
```

A **simple solution** is to use two nested loops. The outer loop is used to pick a starting point and inner loop is used to pick an ending point. For every pair of starting-ending points, we count distinct elements in it and update result if current window is smaller. We use hashing to count distinct elements in a range.

C++

```
 // CPP program to find minimum range that
// contains exactly k distinct numbers.
#include <bits/stdc++.h>
using namespace std;

// Prints the minimum range that contains exactly
```

```
// k distinct numbers.
void minRange(int arr[], int n, int k)
{
    int l = 0, r = n;

    // Consider every element as starting
    // point.
    for (int i = 0; i < n; i++) {

        // Find the smallest window starting
        // with arr[i] and containing exactly
        // k distinct elements.
        unordered_set<int> s;
        int j;
        for (j = i; j < n; j++) {
            s.insert(arr[j]);
            if (s.size() == k) {
                if ((j - i) < (r - l)) {
                    r = j;
                    l = i;
                }
                break;
            }
        }

        // There are less than k distinct elements
        // now, so no need to continue.
        if (j == n)
            break;
    }

    // If there was no window with k distinct
    // elements (k is greater than total distinct
    // elements)
    if (l == 0 && r == n)
        cout << "Invalid k";
    else
        cout << l << " " << r;
}

// Driver code for above function.
int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    minRange(arr, n, k);
    return 0;
```

```
}
```

Java

```
 // Java program to find minimum
// range that contains exactly
// k distinct numbers.
import java.util.*;

class GFG
{

// Prints the minimum range
// that contains exactly k
// distinct numbers.
static void minRange(int arr[],
                     int n, int k)
{
    int l = 0, r = n;

    // Consider every element
    // as starting point.
    for (int i = 0; i < n; i++)
    {

        // Find the smallest window
        // starting with arr[i] and
        // containing exactly k
        // distinct elements.
        Set<Integer> s = new HashSet<Integer>();
        int j;
        for (j = i; j < n; j++)
        {
            s.add(arr[j]);
            if (s.size() == k)
            {
                if ((j - i) < (r - l))
                {
                    r = j;
                    l = i;
                }
                break;
            }
        }

        // There are less than k
        // distinct elements now,
        // so no need to continue.
```

```
        if (j == n)
            break;
    }

    // If there was no window
    // with k distinct elements
    // (k is greater than total
    // distinct elements)
    if (l == 0 && r == n)
        System.out.println("Invalid k");
    else
        System.out.println(l + " " + r);
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = arr.length;
    int k = 3;
    minRange(arr, n, k);
}
}

// This code is contributed
// by Kirti_Mangal
```

Output:

```
0 2
```

Time Complexity : $O(n^2)$

Optimization over above simple solution. The idea is to remove repetitions on left side after we find k distinct elements.

C++

```
 // CPP program to find minimum range that
// contains exactly k distinct numbers.
#include <bits/stdc++.h>
using namespace std;

// prints the minimum range that contains exactly
// k distinct numbers.
```

```
void minRange(int arr[], int n, int k)
{
    // Initially left and right side is -1 and -1,
    // number of distinct elements are zero and
    // range is n.
    int l = 0, r = n;

    int j = -1; // Initialize right side
    map<int, int> hm;
    for (int i=0; i<n; i++)
    {
        while (j < n)
        {
            // increment right side.
            j++;

            // if number of distinct elements less
            // than k.
            if (hm.size() < k)
                hm[arr[j]]++;

            // if distinct elements are equal to k
            // and length is less than previous length.
            if (hm.size() == k && ((r - l) >= (j - i)))
            {
                l = i;
                r = j;
                break;
            }
        }

        // if number of distinct elements less
        // than k, then break.
        if (hm.size() < k)
            break;

        // if distinct elements equals to k then
        // try to increment left side.
        while (hm.size() == k)
        {

            if (hm[arr[i]] == 1)
                hm.erase(arr[i]);
            else
                hm[arr[i]]--;

            // increment left side.
            i++;
```

```
            // it is same as explained in above loop.
            if (hm.size() == k && (r - l) >= (j - i))
            {
                l = i;
                r = j;
            }
        }
        if (hm[arr[i]] == 1)
            hm.erase(arr[i]);
        else
            hm[arr[i]]--;
    }

    if (l == 0 && r == n)
        cout << "Invalid k" << endl;
    else
        cout << l << " " << r << endl;
}

// Driver code for above function.
int main()
{
    int arr[] = { 1, 1, 2, 2, 3, 3, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    minRange(arr, n, k);
    return 0;
}
```

Output:

```
5 7
```

Time complexity of this solution is O(n). In every nested iteration, we either add an element or remove an element. Every element is inserted and removed at most once.

Improved By : Kirti_Mangal

Source

https://www.geeksforgeeks.org/smallest-subarray-k-distinct-numbers/

Chapter 276

Smallest window that contains all characters of string itself

Smallest window that contains all characters of string itself - GeeksforGeeks

Given a string, find the smallest window length with all distinct characters of the given string. For eg. str = “aabcbcdbca”, then the result would be 4 as of the smallest window will be “dbca” .

Examples:

```
Input  : aabcbcdbca
Output : dcba
Explanation :
dbca of length 4 is the smallest
window with highest number of distinct
characters.

Input : aaab
Output : ab
Explanation :
ab of length 2 is the smallest window
with highest number of distinct characters.
```

Above problem states that we have to find the smallest window that contains all the distinct characters of the given string even if the smallest string contains repeating elements.

For example, in “aabcbcdb”, the smallest string that contains all the characters is “abcbcd”. This problem reduces to Find the smallest window in a string containing all characters of another string.

In that problem we find the smallest window that contains all the characters of given pattern.

1- Count all distinct characters in given string.

2- Now follow the algorithm discussed in below post.
https://www.geeksforgeeks.org/find-the-smallest-window-in-a-string-containing-all-characters-of-another-string/
We basically maintain a window of characters. Whenever the window contains all characters of given string, we shrink the window from left side to remove extra characters and then compare its length with smallest window fount so far.

C++

```
 // C++ program to find the smallest window containing
// all characters of a pattern.
#include<bits/stdc++.h>
using namespace std;

const int MAX_CHARS = 256;

// Function to find smallest window containing
// all distinct characters
string findSubString(string str)
{
    int n = str.length();

    // Count all distinct characters.
    int dist_count = 0;
    bool visited[MAX_CHARS] = {false};
    for (int i=0; i<n; i++)
    {
        if (visited[str[i]] == false)
        {
            visited[str[i]] = true;
            dist_count++;
        }
    }

    // Now follow the algorithm discussed in below
    // post. We basically maintain a window of characters
    // that contains all characters of given string.
    // https://www.geeksforgeeks.org/find-the-smallest-window-in-a-string-containing-all-characte
    int start = 0, start_index = -1, min_len = INT_MAX;

    int count = 0;
    int curr_count[MAX_CHARS] = {0};
    for (int j=0; j<n; j++)
    {
        // Count occurrence of characters of string
        curr_count[str[j]]++;

        // If any distinct character matched,
        // then increment count
```

```
        if (curr_count[str[j]] == 1 )
            count++;

        // if all the characters are matched
        if (count == dist_count)
        {
            // Try to minimize the window i.e., check if
            // any character is occurring more no. of times
            // than its occurrence in pattern, if yes
            // then remove it from starting and also remove
            // the useless characters.
            while (curr_count[str[start]] > 1)
            {
                if (curr_count[str[start]] > 1)
                    curr_count[str[start]]--;
                start++;
            }

            // Update window size
            int len_window = j - start + 1;
            if (min_len > len_window)
            {
                min_len = len_window;
                start_index = start;
            }
        }
    }

    // Return substring starting from start_index
    // and length min_len
    return str.substr(start_index, min_len);
}

// Driver code
int main()
{
    string str = "aabcbcdbca";
    cout << "Smallest window containing all distinct"
            " characters is " << findSubString(str);
    return 0;
}
```

Java

```
 // Java program to find the smallest window containing
// all characters of a pattern.
import java.util.Arrays;
public class GFG {
```

```
static final int MAX_CHARS = 256;

// Function to find smallest window containing
// all distinct characters
static String findSubString(String str)
{
    int n = str.length();

    // Count all distinct characters.
    int dist_count = 0;

    boolean[] visited = new boolean[MAX_CHARS];
    Arrays.fill(visited, false);
    for (int i=0; i<n; i++)
    {
        if (visited[str.charAt(i)] == false)
        {
            visited[str.charAt(i)] = true;
            dist_count++;
        }
    }

    // Now follow the algorithm discussed in below
    // post. We basically maintain a window of characters
    // that contains all characters of given string.
    // https://www.geeksforgeeks.org/find-the-smallest-window-in-a-string-containing-all-cha
    int start = 0, start_index = -1;
    int min_len = Integer.MAX_VALUE;

    int count = 0;
    int[] curr_count =  new int[MAX_CHARS];
    for (int j=0; j<n; j++)
    {
        // Count occurrence of characters of string
        curr_count[str.charAt(j)]++;

        // If any distinct character matched,
        // then increment count
        if (curr_count[str.charAt(j)] == 1 )
            count++;

        // if all the characters are matched
        if (count == dist_count)
        {
            // Try to minimize the window i.e., check if
            // any character is occurring more no. of times
            // than its occurrence in pattern, if yes
```

```
                // then remove it from starting and also remove
                // the useless characters.
                while (curr_count[str.charAt(start)] > 1)
                {
                    if (curr_count[str.charAt(start)] > 1)
                        curr_count[str.charAt(start)]--;
                    start++;
                }

                // Update window size
                int len_window = j - start + 1;
                if (min_len > len_window)
                {
                    min_len = len_window;
                    start_index = start;
                }
            }
        }
        // Return substring starting from start_index
        // and length min_len
        return str.substring(start_index, start_index+min_len);
    }

    // Driver code
    public static void main(String args[])
    {
        String str = "aabcbcdbca";
        System.out.println("Smallest window containing all distinct"
               + " characters is " + findSubString(str));
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
Smallest window containing all distinct characters is dbca
```

Related Article :
Length of the smallest sub-string consisting of maximum distinct characters

Improved By : Debasish Chowdhury 1

Source

https://www.geeksforgeeks.org/smallest-window-contains-characters-string/

Chapter 277

Sort an array according to absolute difference with given value

Sort an array according to absolute difference with given value - GeeksforGeeks

Given an array of n distinct elements and a number x, arrange array elements according to the absolute difference with x, i. e., element having minimum difference comes first and so on.
Note : If two or more elements are at equal distance arrange them in same sequence as in the given array.

Examples :

```
Input : arr[] : x = 7, arr[] = {10, 5, 3, 9, 2}
Output : arr[] = {5, 9, 10, 3, 2}
Explanation:
7 - 10 = 3(abs)
7 - 5 = 2
7 - 3 = 4
7 - 9 = 2(abs)
7 - 2 = 5
So according to the difference with X,
elements are arranged as 5, 9, 10, 3, 2.

Input : x = 6, arr[] = {1, 2, 3, 4, 5}
Output :  arr[] = {5, 4, 3, 2, 1}

Input : x = 5, arr[] = {2, 6, 8, 3}
Output :  arr[] = {6, 3, 2, 8}
```

The idea is to use a self balancing binary search tree. We traverse input array and for every element, we find its difference with x and store the difference as key and element as value in self balancing binary search tree. Finally we traverse the tree and print its inorder traversal which is required output.

C++ Implementation :
In C++, self-balancing-bianry-search-tree is implemented by set, map and multimap. We can't use set here as we have key value pairs (not only keys). We also can't directly use map also as a single key can belong to multiple values and map allows a single value for a key. So we use multimap which stores key value pairs and can have multiple values for a key.

1. Store the values in the multimap with the difference with X as key.
2. In multiimap, the values will be already in sorted order according to key i.e. difference with X because it implements self-balancing-bianry-search-tree internally.
3. Update all the values of array with the values of map so that array has the required output.

```
 // C++ program to sort an array according absolute
// difference with x.
#include<bits/stdc++.h>
using namespace std;

// Function to sort an array according absolute
// difference with x.
void rearrange(int arr[], int n, int x)
{
    multimap<int, int> m;
    multimap<int ,int >:: iterator it;
    // Store values in a map with the difference
    // with X as key
    for (int i = 0 ; i < n; i++)
        m.insert(make_pair(abs(x-arr[i]),arr[i]));

    // Update the values of array
    int i = 0;
    for (it = m.begin(); it != m.end(); it++)
        arr[i++] = (*it).second ;
}

// Function to print the array
void printArray(int arr[] , int n)
{
    for (int i = 0 ; i < n; i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
```

```
{
    int arr[] = {10, 5, 3, 9 ,2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 7;
    rearrange(arr, n, x);
    printArray(arr, n);
    return 0;
}
```

Output:

```
5 9 10 3 2
```

Time Complexity : O(n Log n)
Auxiliary Space : O(n)

Improved By : bhaveshsingh

Source

https://www.geeksforgeeks.org/sort-an-array-according-to-absolute-difference-with-given-value/

Chapter 278

Sort elements by frequency | Set 4 (Efficient approach using hash)

Sort elements by frequency | Set 4 (Efficient approach using hash) - GeeksforGeeks

Print the elements of an array in the decreasing frequency if 2 numbers have same frequency then print the one which came first.

Examples:

```
Input : arr[] = {2, 5, 2, 8, 5, 6, 8, 8}
Output : arr[] = {8, 8, 8, 2, 2, 5, 5, 6}

Input : arr[] = {2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8}
Output : arr[] = {8, 8, 8, 2, 2, 5, 5, 6, -1, 9999999}
```

We have discussed different approaches in below posts :
Sort elements by frequency | Set 1
Sort elements by frequency | Set 2
Sorting Array Elements By Frequency | Set 3 (Using STL)

All of the above approaches work in O(n Log n) time where n is total number of elements. In this post, a new approach is discussed that works in **O(n + m Log m)** time where n is total number of elements and m is total number of distinct elements.

The idea is to use hashing.

1. We insert all elements and their counts into a hash. This step takes O(n) time where n is number of elements.

2. We copy contents of hash to an array (or vector) and sort them by counts. This step takes O(m Log m) time where m is total number of distinct elements.
3. For maintaining the order of elements if the frequency is same, we use another hash which has the key as elements of the array and value as the index. If the frequency is same for two elements then sort elements according to the index.

The C++ implementation is given below.

```
 // Sort elements by frequency. If two elements have same
// count, then put the elements that appears first
#include <bits/stdc++.h>
using namespace std;

// Map m2 keeps track of indexes of elements in array
unordered_map<int, int> m2;

// Used for sorting by frequency. And if frequency is same,
// then by appearance
bool sortByVal(const pair<int, int>& a, const pair<int, int>& b)
{
    // If frequency is same then sort by index
    if (a.second == b.second)
        return m2[a.first] < m2[b.first];

    return a.second > b.second;
}

// function to sort elements by frequency
void sortByFreq(int a[], int n)
{
    unordered_map<int, int> m;
    vector<pair<int, int> > v;

    for (int i = 0; i < n; ++i) {

        // Map m is used to keep track of count
        // of elements in array
        m[a[i]]++;

        // Update the value of map m2 only once
        if (m2[a[i]] == 0)
            m2[a[i]] = i + 1;
    }

    // Copy map to vector
    copy(m.begin(), m.end(), back_inserter(v));

    // Sort the element of array by frequency
```

```
    sort(v.begin(), v.end(), sortByVal);

    for (int i = 0; i < v.size(); ++i)
        for (int j = 0; j < v[i].second; ++j)
            cout << v[i].first << " ";
}

// Driver program
int main()
{
    int a[] = { 2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    sortByFreq(a, n);

    return 0;
}
```

Output:

```
8 8 8 2 2 5 5 6 -1 9999999
```

Time Complexity : O(n) + O(m Log m) where n is total number of elements and m is total number of distinct elements

Source

https://www.geeksforgeeks.org/sort-elements-frequency-set-4-efficient-approach-using-hash/

Chapter 279

Sort string of characters

Sort string of characters - GeeksforGeeks

Given a string of lowercase characters from ‘a’ – ‘z’. We need to write a program to print the characters of this string in sorted order.

Examples:

```
Input : bbccdefbbaa
Output : aabbbbccdef

Input : geeksforgeeks
Output : eeeefggkkorss
```

A **simple approach** will be to use sorting algorithms like quick sort or merge sort and sort the input string and print it.

C++

```
 // C++ program to sort a string of characters
#include<bits/stdc++.h>
using namespace std;

// function to print string in sorted order
void sortString(string &str)
{
   sort(str.begin(), str.end());
   cout << str;
}

// Driver program to test above function
int main()
```

```
{
    string s = "geeksforgeeks";
    sortString(s);
    return 0;
}
```

Output:

```
eeeefggkkorss
```

Time Complexity: O(n log n), where n is the length of string.

An **efficient approach** will be to observe first that there can be a total of 26 unique characters only. So, we can store the count of occurrences of all the characters from 'a' to 'z' in a hashed array. The first index of the hashed array will represent character 'a', second will represent 'b' and so on. Finally, we will simply traverse the hashed array and print the characters from 'a' to 'z' the number of times they occurred in input string.

Below is the implementation of above idea:

C++

```
 // C++ program to sort a string of characters
#include<bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

// function to print string in sorted order
void sortString(string &str)
{
    // Hash array to keep count of characters.
    // Initially count of all charters is
    // initialized to zero.
    int charCount[MAX_CHAR] = {0};

    // Traverse string and increment
    // count of characters
    for (int i=0; i<str.length(); i++)

        // 'a'-'a' will be 0, 'b'-'a' will be 1,
        // so for location of character in count
        // array we wil do str[i]-'a'.
        charCount[str[i]-'a']++;

    // Traverse the hash array and print
    // characters
```

```
    for (int i=0;i<MAX_CHAR;i++)
        for (int j=0;j<charCount[i];j++)
            cout << (char)('a'+i);
}

// Driver program to test above function
int main()
{
    string s = "geeksforgeeks";
    sortString(s);
    return 0;
}
```

Java

```
 // Java program to sort
// a string of characters
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // Method to sort a
    // string alphabetically
    public static String sortString(String inputString)
    {
        // convert input
        // string to char array
        char tempArray[] =
             inputString.toCharArray();

        // sort tempArray
        Arrays.sort(tempArray);

        // return new sorted string
        return new String(tempArray);
    }

    // Driver Code
    public static void main(String[] args)
    {
        String inputString = "geeksforgeeks";

        System.out.println(sortString(inputString));
    }
}

// This code is contributed
```

```
// by prabhat kumar singh
```

Output:

```
eeeefggkkorss
```

Time Complexity: O(n), where n is the length of input string.
Auxiliary Space: O(1).

Improved By : prabhat kumar singh

Source

https://www.geeksforgeeks.org/sort-string-characters/

Chapter 280

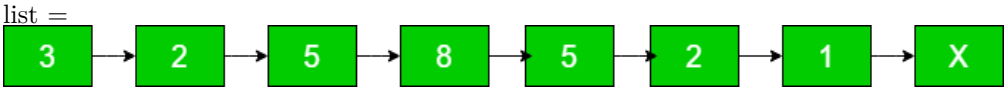
Sort the linked list in the order of elements appearing in the array

Sort the linked list in the order of elements appearing in the array - GeeksforGeeks

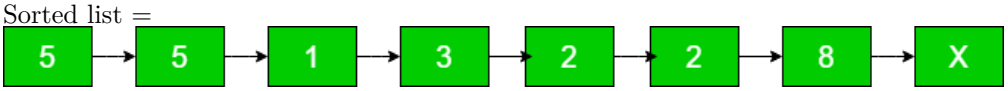
Given an array of size N and a Linked List where elements will be from the array but can also be duplicated, sort the linked list in the order, elements are appearing in the array. It may be assumed that the array covers all elements of the linked list.

arr[] =

list =



Sorted list =



Asked in Amazon

First, make a hash table that stores the frequencies of elements in linked list. Then, simply traverse list and for each element of arr[i] check the frequency in the has table and modify the data of list by arr[i] element upto its frequency and at last Print the list.

```
 // Efficient CPP program to sort given list in order
// elements are appearing in an array
#include <bits/stdc++.h>
using namespace std;
```

```
// Linked list node
struct Node {
    int data;
    struct Node* next;
};

// function prototype for printing the list
void printList(struct Node*);

// Function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;
    new_node -> data = new_data;
    new_node -> next = *head_ref;
    *head_ref = new_node;
}

// function to print the linked list
void printList(struct Node* head)
{
    while (head != NULL) {
        cout << head -> data << " -> ";
        head = head -> next;
    }
}

// Function that sort list in order of apperaing
// elements in an array
void sortlist(int arr[], int N, struct Node* head)
{
    // Store frequencies of elements in a
    // hash table.
    unordered_map<int, int> hash;
    struct Node* temp = head;
    while (temp) {
        hash[temp -> data]++;
        temp = temp -> next;
    }

    temp = head;

    // One by one put elements in lis according
    // to their appearance in array
    for (int i = 0; i < N; i++) {

        // Update 'frequency' nodes with value
```

```
        // equal to arr[i]
        int frequency = hash[arr[i]];
        while (frequency--) {

            // Modify list data as element
            // appear in an array
            temp -> data = arr[i];
            temp = temp -> next;
        }
    }
}

// Driver Code
int main()
{
    struct Node* head = NULL;
    int arr[] = { 5, 1, 3, 2, 8 };
    int N = sizeof(arr) / sizeof(arr[0]);

    // creating the linked list
    push(&head, 3);
    push(&head, 2);
    push(&head, 5);
    push(&head, 8);
    push(&head, 5);
    push(&head, 2);
    push(&head, 1);

    // Function call to sort the list in order
    // elements are apperaing in an array
    sortlist(arr, N, head);

    // print the modified linked list
    cout << "Sorted List:" << endl;
    printList(head);
    return 0;
}
```

Output :

```
Sort list:
5 -> 5 -> 1 -> 3 -> 2 -> 2 -> 8
```

Source

https://www.geeksforgeeks.org/sort-linked-list-order-elements-appearing-array/

Chapter 281

Sorting using trivial hash function

Sorting using trivial hash function - GeeksforGeeks

Sorting using trivial hash function.

Examples:

```
Input :  9 4 3 5 8
Output : 3 4 5 8 9
```

We have read about various sorting algorithms such as heap sort, bubble sort, merge sort and others.
Here we will see how can we sort N elements using hash array. But this algorithm has a limitation. We can sort only those N elements, where the value of elements is not large (typically not above 10^6).

Explanation of sorting using hash:

Step 1: create a hash array of size(max_element), since that is the maximum we will need
Step 2: traverse through all the elements and keep a count of number of occurrence of a particular element.
Step 3: after keep a count of occurrence of all elements in the hash table, simply iterate from 0 to max_element in the hash array
Step 4: while iterating in the hash array, if we find the value stored at any hash position is more then 0, which indicated that the element is present at least once in the original list of elements.
Step 5: Hash[i] has the count of the number of times a element is present in the list, so when its >0, we print those number of times the element.

If you want to store the elements, use another array to store them in a sorted way.
If we want to sort it in a descending order, we simply traverse from max to 0, and repeat the same procedure.

Below is the c++ implementation of the above approach:

```
 // C++ program to sort an array using hash
// function
#include <bits/stdc++.h>
using namespace std;

void sortUsingHash(int a[], int n)
{
    // find the maximum element
    int max = *std::max_element(a, a + n);

    // create a hash function upto the max size
    int hash[max + 1] = { 0 };

    // traverse through all the elements and
    // keep a count
    for (int i = 0; i < n; i++)
        hash[a[i]] += 1;

    // Traverse upto all elements and check if
    // it is present or not. If it is present,
    // then print the element the number of times
    // it's present. Once we have printed n times,
    // that means we have printed n elements
    // so break out of the loop
    for (int i = 0; i <= max; i++) {

        // if present
        if (hash[i]) {

            // print the element that number of
            // times it's present
            for (int j = 0; j < hash[i]; j++) {
                cout << i << " ";
            }
        }
    }
}

// driver program
int main()
{
    int a[] = { 9, 4, 3, 2, 5, 2, 1, 0, 4,
                3, 5, 10, 15, 12, 18, 20, 19 };
    int n = sizeof(a) / sizeof(a[0]);

    sortUsingHash(a, n);
```

```
    return 0;
}
```

Output:

```
0 1 2 2 3 3 4 4 5 5 9 10 12 15 18 19 20
```

How to handle negative numbers?
In case the array has **negative numbers** and **positive** numbers, we keep two hash arrays to keep a track of positive and negative elements.

Explanation of sorting using hashing if the array has negative and positive numbers:
Step 1: Create two hash arrays, one for positive and the other for negative
Step 2: the positive hash array will have a size of max and the negative array will have a size of min
Step 3: traverse from min to 0 in the negative hash array, and print the elements in the same way we did for positives.
Step 4: Traverse from 0 to max for positive elements and print them in the same manner as explained above.

Below is the c++ implementation of the above approach:

```
 // C++ program to sort an array using hash
// function with negative values allowed.
#include <bits/stdc++.h>
using namespace std;

void sortUsingHash(int a[], int n)
{
    // find the maximum elemnet
    int max = *std::max_element(a, a + n);
    int min = abs(*std::min_element(a, a + n));

    // create a hash function upto the max size
    int hashpos[max + 1] = { 0 };
    int hashneg[min + 1] = { 0 };

    // traverse through all the elements and
    // keep a count
    for (int i = 0; i < n; i++) {
        if (a[i] >= 0)
            hashpos[a[i]] += 1;
        else
            hashneg[abs(a[i])] += 1;
    }

    // Traverse up to all negative elements and
```

```
    // check if it is present or not. If it is
    // present, then print the element the number
    // of times it's present. Once we have printed
    // n times, that means we have printed n elements
    // so break out of the loop
    for (int i = min; i > 0; i--) {
        if (hashneg[i]) {

            // print the element that number of times
            // it's present. Print the negative element
            for (int j = 0; j < hashneg[i]; j++) {
                cout << (-1) * i << " ";
            }
        }
    }

    // Traverse upto all elements and check if it is
    // present or not. If it is present, then print
    // the element the number of times it's present
    // once we have printed n times, that means we
    // have printed n elements, so break out of the
    // loop
    for (int i = 0; i <= max; i++) {

        // if present
        if (hashpos[i]) {

            // print the element that number of times
            // it's present
            for (int j = 0; j < hashpos[i]; j++) {
                 cout << i << " ";
            }
        }
    }
}

// driver program to test the above function
int main()
{
    int a[] = { -1, -2, -3, -4, -5, -6, 8, 7,
                        5, 4, 3, 2, 1, 0 };
    int n = sizeof(a) / sizeof(a[0]);
    sortUsingHash(a, n);
    return 0;
}
```

Output:

```
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 7 8
```

Complexity:
This sort function can have complexity O(max_element). So performance depends on that set of data provided.

Limitations:
1. Can only sort array elements of limited range (typically from -10^6 to +10^6)
2. Auxiliary space in worst cases is O(max_element) + O(min_element)

Source

https://www.geeksforgeeks.org/sorting-using-trivial-hash-function/

Chapter 282

Split array to three subarrays such that sum of first and third subarray is equal and maximum

Split array to three subarrays such that sum of first and third subarray is equal and maximum - GeeksforGeeks

Given an array of N integers, the task is to print the sum of the first subarray by splitting the array into exactly three subarrays such that the sum of the first and third subarray elements are equal and the maximum.

Note: All the elements must belong to a subarray and the subarrays can also be empty.

Examples:

> **Input:** a[] = {1, 3, 1, 1, 4}
> **Output:** 5
> Split the N numbers to [1, 3, 1], [] and [1, 4]
>
> **Input:** a[] = {1, 3, 2, 1, 4}
> **Output:** 4
> Split the N numbers to [1, 3], [2, 1] and [4]

A **naive approach** is to check for all possible partitions and use the prefix-sum concept to find out the partitions. The partition which gives the maximum sum of the first subarray will be the answer.

An **efficient approach** is as follows:

- Store the prefix sum and suffix sum of the N numbers.
- Hash the suffix sum's index using a unordered_map in C++ or Hash-map in Java.
- Iterate from the beginning of the array, and check if the prefix sum exists in the suffix array beyond the current index i.

- If it does, then check for the previous maximum value and update accordingly.

Below is the implementation of the above approach:

```
 // C++ program for Split the array into three
// subarrays such that summation of first
// and third subarray is equal and maximum
#include <bits/stdc++.h>
using namespace std;

// Fucntion to return the sum of
// the first subarray
int sumFirst(int a[], int n)
{
    unordered_map<int, int> mp;
    int suf = 0;

    // calculate the suffix sum
    for (int i = n - 1; i >= 0; i--) {
        suf += a[i];
        mp[suf] = i;
    }

    int pre = 0;
    int maxi = -1;

    // iterate from beginning
    for (int i = 0; i < n; i++) {

        // prefix sum
        pre += a[i];

        // check if it exists beyond i
        if (mp[pre] > i) {

            // if greater then previous
            // then update maximum
            if (pre > maxi) {
                maxi = pre;
            }
        }
    }

    // First and second subarray empty
    if (maxi == -1)
        return 0;

    // partition done
```

```
    else
        return maxi;
}

// Driver Code
int main()
{
    int a[] = { 1, 3, 2, 1, 4 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << sumFirst(a, n);
    return 0;
}
```

Output:

```
4
```

Source

https://www.geeksforgeeks.org/split-array-to-three-subarrays-such-that-sum-of-first-and-third-subarray-is-equal-a

Chapter 283

Subarray with no pair sum divisible by K

Subarray with no pair sum divisible by K - GeeksforGeeks

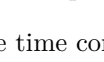
Given an array of N non-negative integers, task is to find the maximum size of a subarray such that the pairwise sum of the elements of this subarray is not divisible by a given integer, K. Also, print this subarray as well. If there are two or more subarrays which follow the above stated condition, then print the first one from the left.

Prerequisite : Subset with no pair sum divisible by K

Examples :

```
Input : arr[] = [3, 7, 1, 9, 2]
        K = 3
Output : 3
         [3, 7, 1]
3 + 7 = 10, 3 + 1 = 4, 7 + 1 = 8, all are
not divisible by 3.
It is not possible to get a subarray of size bigger
than 3 with the above-mentioned property.
[7, 1, 9] is also of the same size but [3, 7, 1] comes first.

Input : arr[] = [2, 4, 4, 3]
        K = 4
Output : 2
         [2, 4]
2 + 4 = 6 is not divisible by 4.
It is not possible to get a subarray of size bigger
than 2 with the above-mentioned property.
[4, 3] is also of the same size but [2, 4] comes first.
```

Naive Approach :
The naive method would be to consider all the subarrays. While considering a subarray, take elements pairwise and compute the sum of the two elements of the pair. If the computed sum is divisible by K, then ignore this subarray and continue with the next subarray. Else, compute the sum of other pairs of this subarray in a similar fashion. If no pair's sum is a multiple of K, then compare the size of this subarray with the maximum size obtained so far and update if required.

The time complexity of this method would be O().

Efficient Approach(Using Hashing) :
We create an empty hash table and insert arr[0] % k into it. Now we traverse remaining elements and maintain a window such that no pair in the window is divisible by k. For every traversed element, we remove starting elements while there exist an element in current window which makes a divisible pair with current element. To check if there is an element in current window, we check if following.
1) If there is an element x such that (K – x % K) is equal to arr[i] % K
2) OR arr[i] % k is 0 and it exists in the hash.
Once we make sure that all elements which can make a pair with arr[i] are removed, we add arr[i] to current window and check if size of current window is more than the maximum window so far.

C++

```
 // CPP code to find the subarray with
// no pair sum divisible by K
#include<bits/stdc++.h>
using namespace std;

// function to find the subarray with
// no pair sum divisible by k
void subarrayDivisibleByK(int arr[], int n, int k)
{
    // hash table to store the remainders
    // obtained on dividing by K
    map<int,int> mp;

    // s : starting index of the
    // current subarray, e : ending
    // index of the current subarray, maxs :
    // starting index of the maximum
    // size subarray so far, maxe : ending
    // index of the maximum size subarray
    // so far
    int s = 0, e = 0, maxs = 0, maxe = 0;

    // insert the first element in the set
    mp[arr[0] % k]++;
```

```
    for (int i = 1; i < n; i++)
    {
        int mod = arr[i] % k;

        // Removing starting elements of current
        // subarray while there is an element in
        // set which makes a pair with mod[i] such
        // that the pair sum is divisible.
        while (mp[k - mod] != 0 ||
               (mod == 0 && mp[mod] != 0))
        {
            mp[arr[s] % k]--;
            s++;
        }

        // include the current element in
        // the current subarray the ending
        // index of the current subarray
        // increments by one
        mp[mod]++;
        e++;

        // compare the size of the current
        // subarray with the maximum size so
        // far
        if ((e - s) > (maxe - maxs))
        {
            maxe = e;
            maxs = s;
        }

    }

    cout << "The maximum size is "
         << maxe - maxs + 1 << " and "
         "the subarray is as follows\n";

    for (int i=maxs; i<=maxe; i++)
        cout << arr[i] << " ";
}

int main()
{
    int k = 3;
    int arr[] = {5, 10, 15, 20, 25};
    int n = sizeof(arr)/sizeof(arr[0]);
    subarrayDivisibleByK(arr, n, k);
```

```
    return 0;
}
```

Java

```
 // Java Program to find the subarray with
// no pair sum divisible by K
import java.io.*;
import java.util.*;

public class GFG {

    // function to find the subarray with
    // no pair sum divisible by k
    static void subarrayDivisibleByK(int []arr,
                                int n, int k)
    {

        // hash table to store the remainders
        // obtained on dividing by K
        int []mp = new int[1000];

        // s : starting index of the
        // current subarray, e : ending
        // index of the current subarray, maxs :
        // starting index of the maximum
        // size subarray so far, maxe : ending
        // index of the maximum size subarray
        // so far
        int s = 0, e = 0, maxs = 0, maxe = 0;

        // insert the first element in the set
        mp[arr[0] % k]++;

        for (int i = 1; i < n; i++)
        {
            int mod = arr[i] % k;

            // Removing starting elements of current
            // subarray while there is an element in
            // set which makes a pair with mod[i] such
            // that the pair sum is divisible.
            while (mp[k - mod] != 0 ||
                (mod == 0 && mp[mod] != 0))
            {
                mp[arr[s] % k]--;
                s++;
            }
```

```
            // include the current element in
            // the current subarray the ending
            // index of the current subarray
            // increments by one
            mp[mod]++;
            e++;

            // compare the size of the current
            // subarray with the maximum size so
            // far
            if ((e - s) > (maxe - maxs))
            {
                maxe = e;
                maxs = s;
            }

        }

        System.out.print("The maximum size is "
                            + (maxe - maxs + 1)
        + " and the subarray is as follows\n");

        for (int i = maxs; i <= maxe; i++)
            System.out.print(arr[i] + " ");
    }

    // Driver Code
    public static void main(String args[])
    {
        int k = 3;
        int []arr = {5, 10, 15, 20, 25};
        int n = arr.length;
        subarrayDivisibleByK(arr, n, k);
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
 # Python3 Program to find the subarray with
# no pair sum divisible by K

# function to find the subarray with
# no pair sum divisible by k
def subarrayDivisibleByK(arr, n, k) :
```

```
# hash table to store the remainders
# obtained on dividing by K
mp = [0] * 1000

# s : starting index of the
# current subarray, e : ending
# index of the current subarray, maxs :
# starting index of the maximum
# size subarray so far, maxe : ending
# index of the maximum size subarray
# so far
s = 0; e = 0; maxs = 0; maxe = 0;

# insert the first element in the set
mp[arr[0] % k] = mp[arr[0] % k] + 1;

for i in range(1, n):
    mod = arr[i] % k

    # Removing starting elements of current
    # subarray while there is an element in
    # set which makes a pair with mod[i] such
    # that the pair sum is divisible.
    while (mp[k - mod] != 0 or (mod == 0
                        and mp[mod] != 0)) :
        mp[arr[s] % k] = mp[arr[s] % k] - 1
        s = s + 1

    # include the current element in
    # the current subarray the ending
    # index of the current subarray
    # increments by one
    mp[mod] = mp[mod] + 1
    e = e + 1

    # compare the size of the current
    # subarray with the maximum size so
    # far
    if ((e - s) > (maxe - maxs)) :
        maxe = e
        maxs = s

print ("The maximum size is {} and the "
               " subarray is as follows"
               .format((maxe - maxs + 1)))

for i in range(maxs, maxe + 1) :
```

```
        print ("{} ".format(arr[i]), end="")

# Driver Code
k = 3
arr = [5, 10, 15, 20, 25]
n = len(arr)
subarrayDivisibleByK(arr, n, k)

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# Program to find the subarray with
// no pair sum divisible by K
using System;
using System.Collections;

class GFG {

    // function to find the subarray with
    // no pair sum divisible by k
    static void subarrayDivisibleByK(int []arr,
                                int n, int k)
    {

        // hash table to store the remainders
        // obtained on dividing by K
        int []mp = new int[1000];

        // s : starting index of the
        // current subarray, e : ending
        // index of the current subarray, maxs :
        // starting index of the maximum
        // size subarray so far, maxe : ending
        // index of the maximum size subarray
        // so far
        int s = 0, e = 0, maxs = 0, maxe = 0;

        // insert the first element in the set
        mp[arr[0] % k]++;

        for (int i = 1; i < n; i++)
        {
            int mod = arr[i] % k;

            // Removing starting elements of current
            // subarray while there is an element in
```

```
            // set which makes a pair with mod[i] such
            // that the pair sum is divisible.
            while (mp[k - mod] != 0 ||
                (mod == 0 && mp[mod] != 0))
            {
                mp[arr[s] % k]--;
                s++;
            }

            // include the current element in
            // the current subarray the ending
            // index of the current subarray
            // increments by one
            mp[mod]++;
            e++;

            // compare the size of the current
            // subarray with the maximum size so
            // far
            if ((e - s) > (maxe - maxs))
            {
                maxe = e;
                maxs = s;
            }

        }

        Console.Write("The maximum size is " +
                            (maxe - maxs + 1) +
            " and the subarray is as follows\n");

        for (int i = maxs; i <= maxe; i++)
            Console.Write(arr[i] + " ");
    }

    // Driver Code
    public static void Main()
    {
        int k = 3;
        int []arr = {5, 10, 15, 20, 25};
        int n = arr.Length;
        subarrayDivisibleByK(arr, n, k);
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// PHP Program to find the
// subarray with no pair
// sum divisible by K

// function to find the subarray
// with no pair sum divisible by k
function subarrayDivisibleByK($arr, $n, $k)
{
    // hash table to store the remainders
    // obtained on dividing by K
    $mp = array_fill(0, 1000, 0);

    // s : starting index of the
    // current subarray, e : ending
    // index of the current subarray, maxs :
    // starting index of the maximum
    // size subarray so far, maxe : ending
    // index of the maximum size subarray
    // so far
    $s = 0;
    $e = 0;
    $maxs = 0;
    $maxe = 0;

    // insert the first
    // element in the set
    $mp[$arr[0] % $k]++;

    for ($i = 1; $i < $n; $i++)
    {
        $mod = $arr[$i] % $k;

        // Removing starting elements
        // of current subarray while
        // there is an element in set
        // which makes a pair with
        // mod[i] such that the pair
        // sum is divisible.
        while ($mp[$k - $mod] != 0 ||
                        ($mod == 0 &&
                         $mp[$mod] != 0))
        {
            $mp[$arr[$s] % $k]--;
            $s++;
        }
```

```
        // include the current element in
        // the current subarray the ending
        // index of the current subarray
        // increments by one
        $mp[$mod]++;
        $e++;

        // compare the size of the current
        // subarray with the maximum size so
        // far
        if (($e - $s) > ($maxe - $maxs))
        {
            $maxe = $e;
            $maxs = $s;
        }

    }

    echo ("The maximum size is ".
              ($maxe - $maxs + 1).
          " and the subarray is".
                " as follows\n");

    for ($i = $maxs; $i <= $maxe; $i++)
        echo ($arr[$i]." ");
}

// Driver Code
$k = 3;
$arr = array(5, 10, 15, 20, 25);
$n = count($arr);
subarrayDivisibleByK($arr, $n, $k);

// This code is contributed by
// Manish Shaw (manishshaw1)
?>
```

Output :

```
The maximum size is 2 and the subarray is as follows
10 15
```

Time Complexity : O(n)

Improved By : aganjali10, manishshaw1

Source

https://www.geeksforgeeks.org/subarray-no-pair-sum-divisible-k/

Chapter 284

Subarrays with distinct elements

Subarrays with distinct elements - GeeksforGeeks

Given an array, the task is to calculate the sum of lengths of contiguous subarrays having all elements distinct.

Examples:

```
Input :  arr[] = {1, 2, 3}
Output : 10
{1, 2, 3} is a subarray of length 3 with
distinct elements. Total length of length
three = 3.
{1, 2}, {2, 3} are 2 subarray of length 2
with distinct elements. Total length of
lengths two = 2 + 2 = 4
{1}, {2}, {3} are 3 subarrays of length 1
with distinct element. Total lengths of
length one = 1 + 1 + 1 = 3
Sum of lengths = 3 + 4 + 3 = 10

Input :  arr[] = {1, 2, 1}
Output : 7

Input :  arr[] = {1, 2, 3, 4}
Output : 20
```

A **simple solution** is to consider all subarrays and for every subarray check if it has distinct elements or not using hashing. And add lengths of all subarrays having distinct elements.

If we use hashing to find distinct elements, then this approach takes $O(n^2)$ time under the assumption that hashing search and insert operations take O(1) time.

An **efficient solution** is based on the fact that if we know all elements in a subarray arr[i..j] are distinct, sum of all lengths of distinct element subarrays in this sub array is ((j-i+1)*(j-i+2))/2. How? the possible lengths of subarrays are 1, 2, 3,……, j – i +1. So, the sum will be ((j – i +1)*(j – i +2))/2.

We first find largest subarray (with distinct elements) starting from first element. We count sum of lengths in this subarray using above formula. For finding next subarray of the distinct element, we increment starting point, i and ending point, j unless (i+1, j) are distinct. If not possible, then we increment i again and move forward the same way.

Below is C++ implementation of this approach:

```
 // C++ program to calculate sum of lengths of subarrays
// of distinct elements.
#include<bits/stdc++.h>
using namespace std;

// Returns sum of lengths of all subarrays with distinct
// elements.
int sumoflength(int arr[], int n)
{
    // For maintaining distinct elements.
    unordered_set<int> s;

    // Initialize ending point and result
    int j = 0, ans = 0;

    // Fix starting point
    for (int i=0; i<n; i++)
    {
        // Find ending point for current subarray with
        // distinct elements.
        while (j < n && s.find(arr[j]) == s.end())
        {
            s.insert(arr[j]);
            j++;
        }

        // Calculating and adding all possible length
        // subarrays in arr[i..j]
        ans += ((j - i) * (j - i + 1))/2;

        // Remove arr[i] as we pick new stating point
        // from next
        s.erase(arr[i]);
    }
```

```
    return ans;
}

// Driven Code
int main()
{
    int arr[] = {1, 2, 3, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << sumoflength(arr, n) << endl;
    return 0;
}
```

Output:

```
20
```

Time Complexity of this solution is O(n). Note that the inner loop runs n times in total as j goes from 0 to n across all outer loops. So we do O(2n) operations which is same as O(n).

Source

https://www.geeksforgeeks.org/subarrays-distinct-elements/

Chapter 285

Sudo Placement | Beautiful Pairs

Sudo Placement | Beautiful Pairs - GeeksforGeeks

Given two arrays of integers where maximum size of first array is big and that of second array is small. Your task is to find if there is a pair in the first array whose sum is present in the second array.

Examples:

```
Input:
4
1 5 10 8
3
2 20 13
Output: 1
```

Approach : We have x + y = z. This can be rewritten as x = z – y. This means, we need to find an element x in array 1 such that it is the result of z(second array) – y(first array). For this, use hashing to keep track of such element x.

```
 // CPP code for finding required pairs
#include <bits/stdc++.h>
using namespace std;

// The function to check if beautiful pair exists
bool pairExists(int arr1[], int m, int arr2[], int n)
{
    // Set for hashing
    unordered_set<int> s;
```

```
    // Traversing the first array
    for (int i = 0; i < m; i++) {

        // Traversing the second array to check for
        // every j corresponding to single i
        for (int j = 0; j < n; j++) {

            // x + y = z => x = y - z
            if (s.find(arr2[j] - arr1[i]) != s.end())

                // if such x exists then we return true
                return true;

        }

       // hash to make use of it next time
       s.insert(arr1[i]);
 }

    // no pair exists
    return false;
}

// Driver Code
int main()
{
    int arr1[] = { 1, 5, 10, 8 };
    int arr2[] = { 2, 20, 13 };

    // If pair exists then 1 else 0
    // 2nd argument as size of first array
    // fourth argument as sizeof 2nd array
    if (pairExists(arr1, 4, arr2, 3))
        cout << 1 << endl;
    else
        cout << 0 << endl;

    return 0;
}
```

Output:

```
1
```

Source

https://www.geeksforgeeks.org/sudo-placement-beautiful-pairs/

Chapter 286

Sudo Placement[1.5] | Partition

Sudo Placement[1.5] | Partition - GeeksforGeeks

Given an array of positive and negative numbers. The task is to find a partition point such that none of the elements of left array are in the right array. If there are multiple partitions, then find the partition at which the absolute difference between the sum of left array and sum of right array ($|sum_{left} - sum_{right}|$) with respect to the partition point is minimum. In case of multiple points, print the first partition point from left which is (last index of left array and first index of right array)/2 . Consider 1-based indexing. The left and right array on partition must have a minimum of 1 element and maximum of n-1 elements. Print -1 if no partition is possible.

Examples:

> **Input:** a[] = {1, 2, -1, 2, 3}
> **Output:** 1
> Left array = {1, 2, -1, 2}
> Right array = {3}
> Sumleft = 4, Sumright = 3
> Difference = 1 which is the minimum possible
>
> **Input:** a[] = {1, 2, 3, 1}
> **Output:** -1

A **naive approach** will be to traverse left and right from every index and check if the partition is possible or not at that index. If the partition is possible, then check if the absolute difference between the sum of an element of left array and element of right array is less than that of the previous obtained value at the partition. After finding the partition point, greedily find the $|\mathbf{sum_{left} - sum_{right}}|$.

Time Complexity: $O(N^2)$

An **efficient solution** will be to store the last index of every occurring element in a hash-map. Since the element values are large, direct indexing cannot be used. Create a *prefix[]* and *suffix[]* array which stores the prefix sum and suffix sum respectively. Initialize a

variable count as 0. Iterate for all the element in the array. A common point of observation is, while traversing if the present element's(A_i) last nonoccurence is not i itself, then we cannot have a partition in between i and the element's last occurrence. While traversing store the maximum of element's last occurrence as the partition cannot be done till then. Once the count is i itself, we can have a partition, now if there are multiple partitions then choose the min $|sum_{left} - sum_{right}|$.

Note : Use of map instead of unordered_map may cause TLE.

Below is the implementation of the above approach.

```
 // C++ program for SP- partition
#include <bits/stdc++.h>
using namespace std;

// Function to find the partition
void partition(int a[], int n)
{
    unordered_map<long long, long long> mpp;

    // mark the last occurrence of every element
    for (int i = 0; i < n; i++)
        mpp[a[i]] = i;

    // calculate the prefix sum
    long long presum[n];
    presum[0] = a[0];
    for (int i = 1; i < n; i++)
        presum[i] = presum[i - 1] + a[i];

    // calculate the suffix sum
    long long sufsum[n];
    sufsum[n - 1] = a[n - 1];
    for (int i = n - 2; i >= 0; i--) {
        sufsum[i] = sufsum[i + 1] + a[i];
    }

    // Check if partition is possible
    bool possible = false;

    // Stores the absolute differnce
    long long ans = 1e18;

    // stores the last index till
    // which there can not be any partition
    long long count = 0;

    // Stores the partition
    long long index = -1;
```

```
    // Check if partition is possible or not
    // donot check for the last element
    // as partition is not possible
    for (int i = 0; i < n - 1; i++) {

        // takes an element and checks it last occurrence
        // stores the maximum of the last occurrence
        // where partition can be done
        count = max(count, mpp[a[i]]);

        // if partition is possible
        if (count == i) {

            // partition is possible
            possible = true;

            // stores the left array sum
            long long sumleft = presum[i];

            // stores the rigth array sum
            long long sumright = sufsum[i + 1];

            // check if the difference is minimum
            if ((abs(sumleft - sumright)) < ans) {
                ans = abs(sumleft - sumright);
                index = i + 1;
            }
        }
    }

    // is partition is possible or not
    if (possible)
        cout << index << ".5" << endl;
    else
        cout << -1 << endl;
}

// Driver Code-
int main()
{
    int a[] = { 1, 2, -1, 2, 3 };
    int n = sizeof(a) / sizeof(a[0]);

    partition(a, n);
    return 0;
}
```

Time Complexity: O(n) under the assumption that unordered_map search works in O(1) time.

Source

https://www.geeksforgeeks.org/sudo-placement1-5-partition/

Chapter 287

Sum of all elements repeating ‘k’ times in an array

Sum of all elements repeating ’k’ times in an array - GeeksforGeeks

Given an array we have to find the sum of all the elements repeating k times in an array. We need to consider every repeating element just once in the sum.

Examples:

```
Input : arr[] = {2, 3, 9, 9}
            k = 1
Output : 5
2 + 3 = 5

Input : arr[] = {9, 8, 8, 8, 10, 4}
          k = 3
Output : 8
```

One **simple solution** is to use two nested loops to count occurrences of every element. While counting, we need to consider an element only if it is not already considered.

C++

```
 // C++ program find sum of elements that
// appear k times.
#include <bits/stdc++.h>
using namespace std;

// Function to count the sum
int sumKRepeating(int arr[], int n, int k)
{
```

```
    int sum = 0;

    // To keep track of processed elements
    vector<bool> visited(n, false);

    // initializing count equal to zero
    for (int i = 0; i < n; i++) {

        // If arr[i] already processed
        if (visited[i] == true)
            continue;

        // counting occurrences of arr[i]
        int count = 1;
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                count++;
                visited[j] = true;
            }
        }

        if (count == k)
           sum += arr[i];
    }

    return sum;
}

// Driver code
int main()
{
    int arr[] = { 9, 9, 10, 11, 8, 8, 9, 8 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << sumKRepeating(arr, n, k);
    return 0;
}
```

Java

```
 // Java program find sum of
// elements that appear k times.
import java.util.*;

class GFG
{
// Function to count the sum
static int sumKRepeating(int arr[],
```

```
                          int n, int k)
{
    int sum = 0;

    // To keep track of
    // processed elements
    Vector<Boolean> visited = new Vector<Boolean>();

    for (int i = 0; i < n; i++)
    visited.add(false);

    // initializing count
    // equal to zero
    for (int i = 0; i < n; i++)
    {

        // If arr[i] already processed
        if (visited.get(i) == true)
            continue;

        // counting occurrences of arr[i]
        int count = 1;
        for (int j = i + 1; j < n; j++)
        {
            if (arr[i] == arr[j])
            {
                count++;
                visited.add(j, true);
            }
        }

        if (count == k)
        sum += arr[i];
    }

    return sum;
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 9, 9, 10, 11,
                  8, 8, 9, 8 };
    int n = arr.length;
    int k = 3;
    System.out.println(sumKRepeating(arr, n, k));
}
}
```

```
// This code is contributed by Arnab Kundu
```

Output:

```
17
```

Time Complexity : O(n*n)
Auxiliary Space : O(n)

An **efficient solution** is use hashing. We count frequencies of all items. Then we traverse hash table and sum those items whose count of occurrences is k.

C++

```
 // C++ program find sum of elements that
// appear k times.
#include <bits/stdc++.h>
using namespace std;

// Returns sum of k appearing elements.
int sumKRepeating(int arr[], int n, int k)
{
   int sum = 0;

   // Count frequencies of all items
   unordered_map<int, int> mp;
   for (int i=0; i<n; i++)
      mp[arr[i]]++;

   // Sum items with frequencies equal to k.
   for (auto x : mp)
     if (x.second == k)
        sum += x.first;
   return sum;
}

// Driver code
int main()
{
    int arr[] = { 9, 9, 10, 11, 8, 8, 9, 8 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << sumKRepeating(arr, n, k);
    return 0;
}
```

Output:

```
17
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : andrew1234

Source

https://www.geeksforgeeks.org/sum-of-all-elements-repeating-k-times-in-an-array/

Chapter 288

Sum of f(a[i], a[j]) over all pairs in an array of n integers

Sum of f(a[i], a[j]) over all pairs in an array of n integers - GeeksforGeeks

Given an array of n integers, find the sum of f(a[i], a[j]) of all pairs (i, j) such that (1 <= i < j <= n).
f(a[i], a[j]):

```
If |a[j]-a[i]| > 1
   f(a[i], a[j]) = a[j] - a[i]
Else // if |a[j]-a[i]| <= 1
   f(a[i], a[j]) = 0
```

Examples:

```
Input : 6 6 4 4
Output : -8
Explanation:
All pairs are: (6 - 6) + (6 - 6) +
(6 - 6) + (4 - 6) + (4 - 6) + (4 - 6) +
(4 - 6) + (4 - 4) + (4 - 4) = -8

Input: 1 2 3 1 3
Output: 4
Explanation: the pairs that add up are:
(3, 1), (3, 1) to give 4, rest all pairs
according to condition gives 0.
```

A **naive approach** is to iterate through all pairs and calculate f(a[i], a[j]) and summing it while traversing in two nested loops will give us our answer.
Time Complexity: O(n^2)

A **efficient approach** will be to use a map/hash function to keep a count of every occurring numbers and then traverse through the list. While traversing through the list, we multiply the count of numbers that are before it and the number itself. Then subtract this result with the pre-sum of the number before that number to get the sum of difference of all pairs possible with that number. To remove all pairs whose absolute difference is <=1, simply subtract the count of occurrence of (number-1) and (number+1) from the previously computed sum. Here we subtract count of (number-1) from the computed sum as it had been previously added to the sum, and we add (number+1) count since the negative has been added to the pre-computed sum of all pairs.

Time Complexity : O(n)

Below is the implementation of the above approach :

C++

```
 // CPP program to calculate the
// sum of f(a[i], aj])
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the sum
int sum(int a[], int n)
{
    // map to keep a count of occurrences
    unordered_map<int, int> cnt;

    // Traverse in the list from start to end
    // number of times a[i] can be in a pair and
    // to get the difference we subtract pre_sum.
    int ans = 0, pre_sum = 0;
    for (int i = 0; i < n; i++) {
        ans += (i * a[i]) - pre_sum;
        pre_sum += a[i];

        // if the (a[i]-1) is present then
        // subtract that value as f(a[i], a[i]-1)=0
        if (cnt[a[i] - 1])
            ans -= cnt[a[i] - 1];

        // if the (a[i]+1) is present then
        // add that value as f(a[i], a[i]-1)=0
        // here we add as a[i]-(a[i]-1)<0 which would
        // have been added as negative sum, so we add
        // to remove this pair from the sum value
```

```
        if (cnt[a[i] + 1])
            ans += cnt[a[i] + 1];

        // keeping a counter for every element
        cnt[a[i]]++;
    }
    return ans;
}

// Driver code
int main()
{
    int a[] = { 1, 2, 3, 1, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << sum(a, n);
    return 0;
}
```

Java

```
 // Java program to calculate
// the sum of f(a[i], aj])
import java.util.*;
public class GfG {

    // Function to calculate the sum
    public static int sum(int a[], int n)
    {
        // Map to keep a count of occurrences
        Map<Integer,Integer> cnt = new HashMap<Integer,Integer>();

        // Traverse in the list from start to end
        // number of times a[i] can be in a pair and
        // to get the difference we subtract pre_sum
        int ans = 0, pre_sum = 0;
        for (int i = 0; i < n; i++) {
            ans += (i * a[i]) - pre_sum;
            pre_sum += a[i];

            // If the (a[i]-1) is present then subtract
            // that value as f(a[i], a[i]-1) = 0
            if (cnt.containsKey(a[i] - 1))
                ans -= cnt.get(a[i] - 1);

            // If the (a[i]+1) is present then
            // add that value as f(a[i], a[i]-1)=0
            // here we add as a[i]-(a[i]-1)<0 which would
            // have been added as negative sum, so we add
```

```
            // to remove this pair from the sum value
            if (cnt.containsKey(a[i] + 1))
                ans += cnt.get(a[i] + 1);

            // keeping a counter for every element
            if(cnt.containsKey(a[i])) {
                cnt.put(a[i], cnt.get(a[i]) + 1);
            }
            else {
                cnt.put(a[i], 1);
            }
        }
        return ans;
    }

    // Driver code
    public static void main(String args[])
    {
        int a[] = { 1, 2, 3, 1, 3 };
        int n = a.length;
        System.out.println(sum(a, n));
    }
}

// This code is contributed by Swetank Modi
```

Output :

```
4
```

Source

https://www.geeksforgeeks.org/sum-fai-aj-pairs-array-n-integers/

Chapter 289

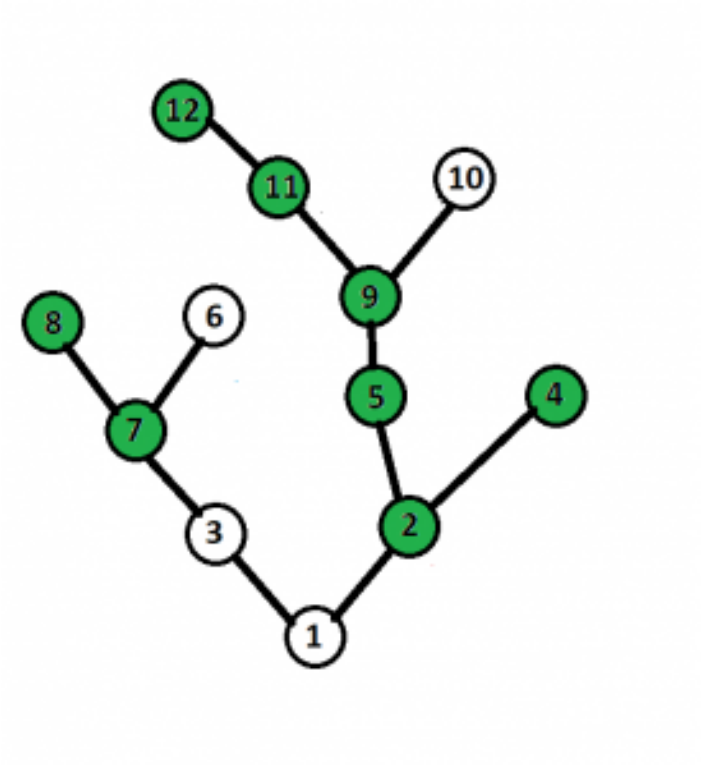
Traversal of tree with k jumps allowed between nodes of same height

Traversal of tree with k jumps allowed between nodes of same height - GeeksforGeeks

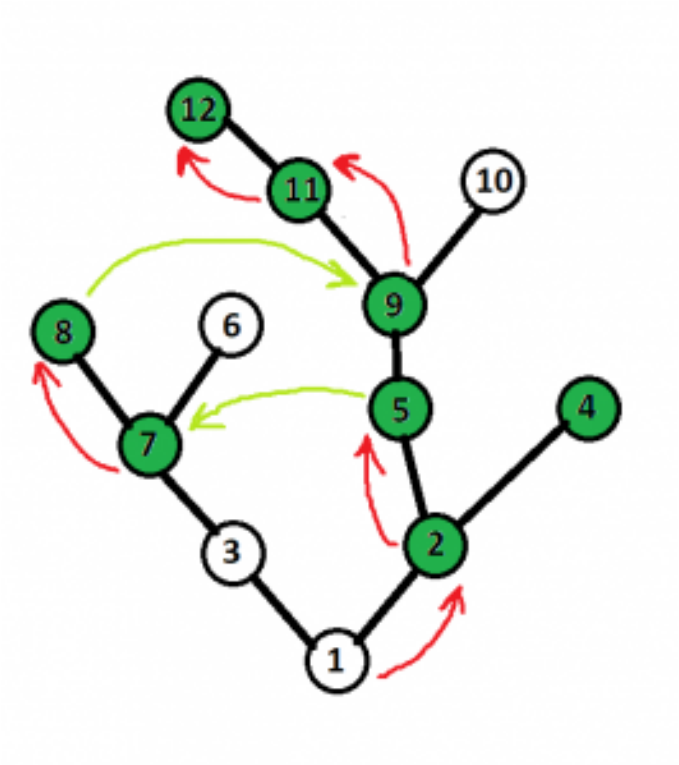
Consider a tree with n nodes and root. You can jump from one node to any other node on the same height a maximum of k times on total jumps. Certain nodes of the tree contain a fruit, find out the maximum number of fruits he can collect.
Example :

```
Input Tree :
Number of Nodes = 12
Number of jumps allowed : 2
Edges:
1 2
1 3
2 4
2 5
5 9
9 10
9 11
11 12
1 3
3 7
7 6
7 8
Nodes Containing Fruits : 2 4 5 7 8 9 11 12
Output: 7
```

Tree for above testcase :



Explanation:



Approach: The idea is to use DFS to create a Height Adjacency List of the Nodes and to store the parents. Then use another dfs to compute the maximum no of special nodes that can be reached using the following dp state:

```
dp[current_node][j] = max( max{ dp[child_i][j], for all children of current_node },
                           max{ dp[node_at_same_height_i][j - 1],
                           for all nodes at same height as current_node} )
```

Thus, **dp[Root_Node][Total_no_of_Jumps]** gives the answer to the problem.

Below is the implementation of above approach :

```
 // Program to demonstrate tree traversal with
// ability to jump between nodes of same height
#include <bits/stdc++.h>
using namespace std;

#define N 1000

vector<int> H[N];

// Arrays declaration
```

```
int Fruit[N];
int Parent[N];
int dp[N][20];

// Function for DFS
void dfs1(vector<int> tree[], int s,
          int p, int h)
{
    Parent[s] = p;
    int i;
    H[h].push_back(s);
    for (i = 0; i < tree[s].size(); i++) {
        int v = tree[s][i];
        if (v != p)
            dfs1(tree, v, s, h + 1);
    }
}

// Function for DFS
int dfs2(vector<int> tree[], int s,
         int p, int h, int j)
{
    int i;
    int ans = 0;
    if (dp[s][j] != -1)
        return dp[s][j];

    // jump
    if (j > 0) {
        for (i = 0; i < H[h].size(); i++) {
            int v = H[h][i];
            if (v != s)
                ans = max(ans, dfs2(tree, v,
                        Parent[v], h, j - 1));
        }
    }

    // climb
    for (i = 0; i < tree[s].size(); i++) {
        int v = tree[s][i];
        if (v != p)
            ans = max(ans, dfs2(tree, v, s, h + 1, j));
    }

    if (Fruit[s] == 1)
        ans++;
    dp[s][j] = ans;
```

```
    return ans;
}

// Function to calculate and
// return maximum number of fruits
int maxFruit(vector<int> tree[],
             int NodesWithFruits[],
             int n, int m, int k)
{
    // reseting dp table and Fruit array
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 20; j++)
            dp[i][j] = -1;
        Fruit[i] = 0;
    }

    // This array is used to mark
    // which nodes contain Fruits
    for (int i = 0; i < m; i++)
        Fruit[NodesWithFruits[i]] = 1;

    dfs1(tree, 1, 0, 0);
    int ans = dfs2(tree, 1, 0, 0, k);

    return ans;
}

// Function to add Edge
void addEdge(vector<int> tree[], int u, int v)
{
    tree[u].push_back(v);
    tree[v].push_back(u);
}

// Driver Code
int main()
{
    int n = 12; // Number of nodes
    int k = 2; // Number of allowed jumps

    vector<int> tree[N];

    // Edges
    addEdge(tree, 1, 2);
    addEdge(tree, 1, 3);
    addEdge(tree, 2, 4);
    addEdge(tree, 2, 5);
    addEdge(tree, 5, 9);
```

```
    addEdge(tree, 9, 10);
    addEdge(tree, 9, 11);
    addEdge(tree, 11, 12);
    addEdge(tree, 1, 3);
    addEdge(tree, 3, 7);
    addEdge(tree, 7, 6);
    addEdge(tree, 7, 8);

    int NodesWithFruits[] = { 2, 4, 5, 7, 8, 9, 11, 12 };

    // Number of nodes with fruits
    int m = sizeof(NodesWithFruits) / sizeof(NodesWithFruits[0]);

    int ans = maxFruit(tree, NodesWithFruits, n, m, k);

    cout << ans << endl;

    return 0;
}
```

Output:

```
7
```

Time Complexity : O(n*n*k) (worst case, eg: 2 level tree with the root having n-1 child nodes)

Source

https://www.geeksforgeeks.org/traversal-tree-ability-jump-nodes-height/

Chapter 290

Union and Intersection of two Linked Lists

Union and Intersection of two Linked Lists - GeeksforGeeks

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elments in output lists doesn't matter.

Example:

```
Input:
   List1: 10->15->4->20
   lsit2:  8->4->2->10
Output:
   Intersection List: 4->10
   Union List: 2->8->20->4->15->10
```

Method 1 (Simple)
Following are simple algorithms to get union and intersection lists respectively.

Intersection (list1, list2)
Initialize result list as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result.

Union *(list1, list2):*
Initialize result list as NULL. Traverse list1 and add all of its elements to the result.
Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

This method assumes that there are no duplicates in the given lists.

Thanks to Shekhu for suggesting this method. Following are C and Java implementations of this method.

C/C++

```
 // C/C++ program to find union and intersection of two unsorted
// linked lists
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the beginning of
   a linked list*/
void push(struct Node** head_ref, int new_data);

/* A utility function to check if given data is present in a list */
bool isPresent(struct Node *head, int data);

/* Function to get union of two linked lists head1 and head2 */
struct Node *getUnion(struct Node *head1, struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Insert all elements of list1 to the result list
    while (t1 != NULL)
    {
        push(&result, t1->data);
        t1 = t1->next;
    }

    // Insert those elements of list2 which are not
    // present in result list
    while (t2 != NULL)
    {
        if (!isPresent(result, t2->data))
            push(&result, t2->data);
        t2 = t2->next;
    }

    return result;
}

/* Function to get intersection of two linked lists
  head1 and head2 */
```

```
struct Node *getIntersection(struct Node *head1,
                             struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1;

    // Traverse list1 and search each element of it in
    // list2. If the element is present in list 2, then
    // insert the element to result
    while (t1 != NULL)
    {
        if (isPresent(head2, t1->data))
            push (&result, t1->data);
        t1 = t1->next;
    }

    return result;
}

/* A utility function to insert a node at the begining of a linked list*/
void push (struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* A utility function to print a linked list*/
void printList (struct Node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
}

/* A utility function that returns true if data is
   present in linked list else return false */
```

```
bool isPresent (struct Node *head, int data)
{
    struct Node *t = head;
    while (t != NULL)
    {
        if (t->data == data)
            return 1;
        t = t->next;
    }
    return 0;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* intersecn = NULL;
    struct Node* unin = NULL;

    /*create a linked lits 10->15->5->20 */
    push (&head1, 20);
    push (&head1, 4);
    push (&head1, 15);
    push (&head1, 10);

    /*create a linked lits 8->4->2->10 */
    push (&head2, 10);
    push (&head2, 2);
    push (&head2, 4);
    push (&head2, 8);

    intersecn = getIntersection (head1, head2);
    unin = getUnion (head1, head2);

    printf ("\n First list is \n");
    printList (head1);

    printf ("\n Second list is \n");
    printList (head2);

    printf ("\n Intersection list is \n");
    printList (intersecn);

    printf ("\n Union list is \n");
    printList (unin);
```

```
    return 0;
}
```

Java

```
 // Java program to find union and intersection of two unsorted
// linked lists
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Function to get Union of 2 Linked Lists */
    void getUnion(Node head1, Node head2)
    {
        Node t1 = head1, t2 = head2;

        //insert all elements of list1 in the result
        while (t1 != null)
        {
            push(t1.data);
            t1 = t1.next;
        }

        // insert those elements of list2 that are not present
        while (t2 != null)
        {
            if (!isPresent(head, t2.data))
                push(t2.data);
            t2 = t2.next;
        }
    }

    void getIntersection(Node head1, Node head2)
    {
        Node result = null;
        Node t1 = head1;
```

```
    // Traverse list1 and search each element of it in list2.
    // If the element is present in list 2, then insert the
    // element to result
    while (t1 != null)
    {
        if (isPresent(head2, t1.data))
            push(t1.data);
        t1 = t1.next;
    }
}

/* Utility function to print list */
void printList()
{
    Node temp = head;
    while(temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}


/*  Inserts a node at start of linked list */
void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
              Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}


/* A utilty function that returns true if data is present
   in linked list  else return false */
boolean isPresent (Node head, int data)
{
    Node t = head;
    while (t != null)
    {
        if (t.data == data)
```

```
                return true;
            t = t.next;
        }
        return false;
    }


    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();
        LinkedList unin = new LinkedList();
        LinkedList intersecn = new LinkedList();

        /*create a linked lits 10->15->5->20 */
        llist1.push(20);
        llist1.push(4);
        llist1.push(15);
        llist1.push(10);

        /*create a linked lits 8->4->2->10 */
        llist2.push(10);
        llist2.push(2);
        llist2.push(4);
        llist2.push(8);

        intersecn.getIntersection(llist1.head, llist2.head);
        unin.getUnion(llist1.head, llist2.head);

        System.out.println("First List is");
        llist1.printList();

        System.out.println("Second List is");
        llist2.printList();

        System.out.println("Intersection List is");
        intersecn.printList();

        System.out.println("Union List is");
        unin.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

Output:

```
 First list is
```

```
10 15 4 20
 Second list is
8 4 2 10
 Intersection list is
4 10
 Union list is
2 8 20 4 15 10
```

Time Complexity: O(mn) for both union and intersection operations. Here m is the number of elements in first list and n is the number of elements in second list.

Method 2 (Use Merge Sort)
In this method, algorithms for Union and Intersection are very similar. First we sort the given lists, then we traverse the sorted lists to get union and intersection.
Following are the steps to be followed to get union and intersection lists.

1) Sort the first Linked List using merge sort. This step takes O(mLogm) time. Refer this post for details of this step.
2) Sort the second Linked List using merge sort. This step takes O(nLogn) time. Refer this post for details of this step.
3) Linearly scan both sorted lists to get the union and intersection. This step takes O(m + n) time. This step can be implemented using the same algorithm as sorted arrays algorithm discussed here.

Time complexity of this method is O(mLogm + nLogn) which is better than method 1's time complexity.

Method 3 (Use Hashing)
Union (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse both lists one by one, for each element being visited, look the element in hash table. If the element is not present, then insert the element to result list. If the element is present, then ignore it.

Intersection (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse list1. For each element being visited in list1, insert the element in hash table. Traverse list2, for each element being visited in list2, look the element in hash table. If the element is present, then insert the element to result list. If the element is not present, then ignore it.

Both of the above methods assume that there are no duplicates.

```
 // Java code for Union and Intersection of two
// Linked Lists
import java.util.HashMap;
import java.util.HashSet;

class LinkedList {
    Node head; // head of list

    /* Linked list Node*/
    class Node
```

```
{
    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }
}

/* Utility function to print list */
void printList()
{
    Node temp = head;
    while(temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}

/* Inserts a node at start of linked list */
void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
    Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}

public void append(int new_data)
{
    if(this.head == null)
    {
        Node n = new Node(new_data);
        this.head = n;
        return;
    }
    Node n1 = this.head;
    Node n2 = new Node(new_data);
    while(n1.next != null)
    {
```

```
            n1 = n1.next;
        }

        n1.next = n2;
        n2.next = null;
    }

    /* A utilty function that returns true if data is
    present in linked list else return false */
    boolean isPresent (Node head, int data)
    {
        Node t = head;
        while (t != null)
        {
            if (t.data == data)
                return true;
            t = t.next;
        }
        return false;
    }

LinkedList getIntersection(Node head1, Node head2)
{
    HashSet<Integer> hset = new HashSet<>();
    Node n1 = head1;
    Node n2 = head2;
    LinkedList result = new LinkedList();

    // loop stores all the elements of list1 in hset
    while(n1 != null)
    {
        if(hset.contains(n1.data))
        {
            hset.add(n1.data);
        }
        else
        {
            hset.add(n1.data);
        }
        n1 = n1.next;
    }

    //For every element of list2 present in hset
    //loop inserts the element into the result
    while(n2 != null)
    {
        if(hset.contains(n2.data))
        {
```

```
            result.push(n2.data);
        }
        n2 = n2.next;
    }
    return result;
}

LinkedList getUnion(Node head1, Node head2)
{
    // HashMap that will store the
    // elements of the lists with their counts
    HashMap<Integer, Integer> hmap = new HashMap<>();
    Node n1 = head1;
    Node n2 = head2;
    LinkedList result = new LinkedList();

    // loop inserts the elements and the count of
    // that element of list1 into the hmap
    while(n1 != null)
    {
        if(hmap.containsKey(n1.data))
        {
            int val = hmap.get(n1.data);
            hmap.put(n1.data, val + 1);
        }
        else
        {
            hmap.put(n1.data, 1);
        }
        n1 = n1.next;
    }

    // loop further adds the elements of list2 with
    // their counts into the hmap
    while(n2 != null)
    {
        if(hmap.containsKey(n2.data))
        {
            int val = hmap.get(n2.data);
            hmap.put(n2.data, val + 1);
        }
        else
        {
            hmap.put(n2.data, 1);
        }
        n2 = n2.next;
    }
```

```
    // Eventually add all the elements
    // into the result that are present in the hmap
    for(int a:hmap.keySet())
    {
        result.append(a);
    }
    return result;
}

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();
        LinkedList union = new LinkedList();
        LinkedList intersection = new LinkedList();

        /*create a linked list 10->15->4->20 */
        llist1.push(20);
        llist1.push(4);
        llist1.push(15);
        llist1.push(10);

        /*create a linked list 8->4->2->10 */
        llist2.push(10);
        llist2.push(2);
        llist2.push(4);
        llist2.push(8);

        intersection = intersection.getIntersection(llist1.head,
                                                    llist2.head);
        union=union.getUnion(llist1.head, llist2.head);

        System.out.println("First List is");
        llist1.printList();

        System.out.println("Second List is");
        llist2.printList();

        System.out.println("Intersection List is");
        intersection.printList();

        System.out.println("Union List is");
        union.printList();
    }
}
// This code is contributed by Kamal Rawal
```

Output:

```
First List is
10 15 4 20
Second List is
8 4 2 10
Intersection List is
10 4
Union List is
2 4 20 8 10 15
```

Time complexity of this method depends on the hashing technique used and the distribution of elements in input lists. In practical, this approach may turn out to be better than above 2 methods.

Source

https://www.geeksforgeeks.org/union-and-intersection-of-two-linked-lists/

Chapter 291

Union and Intersection of two linked lists | Set-2 (Using Merge Sort)

Union and Intersection of two linked lists | Set-2 (Using Merge Sort) - GeeksforGeeks

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter.

Examples:

```
Input:
   List1: 10 -> 15 -> 4 ->20
   lsit2:  8 -> 4 -> 2 -> 10
Output:
   Intersection List: 4 -> 10
   Union List: 2 -> 8 -> 20 -> 4 -> 15 -> 10
```

There were three methods discussed in this post with an implementation of Method 1.
In this post, we will see an implementation of Method 2 i.e. Using Merge sort.

```
Implementation:
Following are the steps to be followed to get union and intersection lists.

1) Sort both Linked Lists using merge sort.
   This step takes O(mLogm) time.

2) Linearly scan both sorted lists to get the union and intersection.
  This step takes O(m + n) time.
```

Just like Method 1, This method also assumes that there are distinct elements in the lists.

```
 // C++ program to find union and intersection of
// two unsorted linked lists in O(n Log n) time.
#include<iostream>
using namespace std;

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the begining
   of a linked list*/
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* UTILITY FUNCTIONS */
/* Split the nodes of the given list into front and back halves,
    and return the two lists using the reference parameters.
    If the length is odd, the extra node should go in the front list.
    Uses the fast/slow pointer strategy. */
void FrontBackSplit(struct Node* source, struct Node** frontRef,
                                         struct Node** backRef)
{
    struct Node* fast;
    struct Node* slow;
    if (source==NULL || source->next==NULL)
    {
        /* length < 2 cases */
        *frontRef = source;
        *backRef = NULL;
    }
```

```
    else
    {
        slow = source;
        fast = source->next;

        /* Advance 'fast' two nodes, and advance 'slow' one node */
        while (fast != NULL)
        {
            fast = fast->next;
            if (fast != NULL)
            {
                slow = slow->next;
                fast = fast->next;
            }
        }

        /* 'slow' is before the midpoint in the list,
            so split it in two at that point. */
        *frontRef = source;
        *backRef = slow->next;
        slow->next = NULL;
    }
}

/* See https://www.geeksforgeeks.org/?p=3622 for details
   of this function */
struct Node* SortedMerge(struct Node* a, struct Node* b)
{
    struct Node* result = NULL;

    /* Base cases */
    if (a == NULL)
        return(b);
    else if (b==NULL)
        return(a);

    /* Pick either a or b, and recur */
    if (a->data <= b->data)
    {
        result = a;
        result->next = SortedMerge(a->next, b);
    }
    else
    {
        result = b;
        result->next = SortedMerge(a, b->next);
    }
    return(result);
```

```
}


/* sorts the linked list by changing next pointers
  (not data) */
void mergeSort(struct Node** headRef)
{
    struct Node *head = *headRef;
    struct Node *a, *b;

    /* Base case -- length 0 or 1 */
    if ((head == NULL) || (head->next == NULL))
        return;

    /* Split head into 'a' and 'b' sublists */
    FrontBackSplit(head, &a, &b);

    /* Recursively sort the sublists */
    mergeSort(&a);
    mergeSort(&b);

    /* answer = merge the two sorted lists together */
    *headRef = SortedMerge(a, b);
}

/* Function to get union of two linked lists head1 and head2 */
struct Node *getUnion(struct Node *head1, struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Traverse both lists and store the  element in
    // the resu1tant list
    while (t1 != NULL && t2 != NULL)
    {
        // Move to the next of first list
        // if its element is smaller
        if (t1->data < t2->data)
        {
            push(&result, t1->data);
            t1 = t1->next;
        }

        // Else move to the next of second list
        else if (t1->data>t2->data)
        {
            push(&result, t2->data);
            t2 = t2->next;
```

```
        }

        // If same then move to the next node
        // in both lists
        else
        {
            push(&result, t2->data);
            t1 = t1->next;
            t2 = t2->next;
        }
    }

    /* Print remaining elements of the lists */
    while (t1 != NULL)
    {
        push(&result, t1->data);
        t1 = t1->next;
    }
    while (t2 != NULL)
    {
        push(&result, t2->data);
        t2 = t2->next;
    }

    return result;
}

/* Function to get intersection of two linked lists
head1 and head2 */
struct Node *getIntersection(struct Node *head1,
                             struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Traverse both lists and store the same element
    // in the resu1tant list
    while (t1 != NULL && t2 != NULL)
    {
        // Move to the next of first list if smaller
        if (t1->data < t2->data)
            t1 = t1->next;

        // Move to the next of second list if it is smaller
        else if (t1->data > t2->data)
            t2 = t2->next;

        // If both are same
```

```
        else
        {
            // Store current element in the list
            push(&result, t2->data);

            // Move to the next node of both lists
            t1 = t1->next;
            t2 = t2->next;
        }
    }

    //return the resultant list
    return result;
}


/* A utility function to print a linked list*/
void printList (struct Node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* intersection_list = NULL;
    struct Node* union_list = NULL;

    /*create a linked lits 11->10->15->4->20 */
    push(&head1, 20);
    push(&head1, 4);
    push(&head1, 15);
    push(&head1, 10);
    push(&head1, 11);

    /*create a linked lits 8->4->2->10 */
    push(&head2, 10);
    push(&head2, 2);
    push(&head2, 4);
    push(&head2, 8);
```

```
    /* Sort the above created Linked List */
    mergeSort(&head1);
    mergeSort(&head2);

    intersection_list = getIntersection(head1, head2);
    union_list = getUnion(head1, head2);

    printf("First list is \n");
    printList(head1);

    printf("\nSecond list is \n");
    printList(head2);

    printf("\nIntersection list is \n");
    printList(intersection_list);

    printf("\nUnion list is \n");
    printList(union_list);

    return 0;
}
```

Output:

```
First list is
4 10 11 15 20
Second list is
2 4 8 10
Intersection list is
10 4
Union list is
20 15 11 10 8 4 2
```

Time complexity of this method is O(m Log m + n Log n).

In the next post, Method-3 will be discussed i.e. using hashing.

Source

https://www.geeksforgeeks.org/union-intersection-two-linked-lists-set-2-using-merge-sort/

Chapter 292

Union and Intersection of two linked lists | Set-3 (Hashing)

Union and Intersection of two linked lists | Set-3 (Hashing) - GeeksforGeeks

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter.

Examples:

```
Input:
   List1: 10 -> 15 -> 4 -> 20
   list2:  8 -> 4 -> 2 -> 10
Output:
   Intersection List: 4 -> 10
   Union List: 2 -> 8 -> 20 -> 4 -> 15 -> 10
```

We have already discussed Method-1 and Method-2 of this question.
In this post, its Method-3 (Using Hashing) is discussed with a Time Complexity of O(m+n) i.e. better than both methods discussed earlier.

```
Implementation:
1- Start traversing both the lists.
   a) Store the current element of both lists
      with its occurrence in the map.
2- For Union: Store all the elements of the map
   in the resultant list.
3- For Intersection: Store all the elements only
   with an occurrence of 2 as 2 denotes that
   they are present in both the lists.
```

Below is C++ implementation of above steps.

```
 // C++ program to find union and intersection of
// two unsorted linked lists in O(m+n) time.
#include<bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the
   begining of a linked list*/
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Utility function to store the elements of both list */
void storeEle(struct Node* head1, struct Node *head2,
                    unordered_map<int, int> &eleOcc)
{
    struct Node* ptr1 = head1;
    struct Node* ptr2 = head2;

    // Traverse both lists
    while (ptr1 != NULL || ptr2 != NULL)
    {
        // store element in the map
        if (ptr1!=NULL)
        {
            eleOcc[ptr1->data]++;
            ptr1=ptr1->next;
        }
```

```
        // store element in the map
        if (ptr2 != NULL)
        {
            eleOcc[ptr2->data]++;
            ptr2=ptr2->next;
        }
    }
}

/* Function to get union of two linked lists head1
   and head2 */
struct Node *getUnion(unordered_map<int, int> eleOcc)
{
    struct Node *result = NULL;

    // Push all the elements into the resultant list
    for (auto it=eleOcc.begin(); it!=eleOcc.end(); it++)
        push(&result, it->first);

    return result;
}

/* Function to get intersection of two linked lists
   head1 and head2 */
struct Node *getIntersection(unordered_map<int, int> eleOcc)
{
    struct Node *result = NULL;

    // Push a node with an element having occurrence
    // of 2 as that means the current element is present
    // in both the lists
    for (auto it=eleOcc.begin(); it!=eleOcc.end(); it++)
        if (it->second == 2)
            push(&result, it->first);

    // return resultant list
    return result;
}

/* A utility function to print a linked list*/
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
```

```
}

// Prints union and intersection of lists with head1
// and head2.
void printUnionIntersection(Node *head1, Node *head2)
{
    // Store all the elements of both lists in the map
    unordered_map<int, int> eleOcc;
    storeEle(head1, head2, eleOcc);

    Node *intersection_list = getIntersection(eleOcc);
    Node *union_list = getUnion(eleOcc);

    printf("\nIntersection list is \n");
    printList(intersection_list);

    printf("\nUnion list is \n");
    printList(union_list);
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    /* create a linked lits 11->10->15->4->20 */
    push(&head1, 1);
    push(&head1, 2);
    push(&head1, 3);
    push(&head1, 4);
    push(&head1, 5);

    /* create a linked lits 8->4->2->10 */
    push(&head2, 1);
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 6);

    printf("First list is \n");
    printList(head1);

    printf("\nSecond list is \n");
    printList(head2);

    printUnionIntersection(head1, head2);
```

```
    return 0;
}
```

Output:

```
First list is
5 4 3 2 1
Second list is
6 5 3 1
Intersection list is
3 5 1
Union list is
3 4 6 5 2 1
```

We can also handle the case of duplicates by maintaining separate Hash for both the lists.

Time Complexity : O(m + n)
Auxiliary Space : O(m + n)

Source

https://www.geeksforgeeks.org/union-intersection-two-linked-lists-set-3-hashing/

Chapter 293

Unique element in an array where all elements occur k times except one

Unique element in an array where all elements occur k times except one - GeeksforGeeks

Given an array which contains all elements occurring k times, but one occurs only once. Find that unique element.

Examples:

```
Input  : arr[] = {6, 2, 5, 2, 2, 6, 6}
            k = 3
Output : 5
Every element appears 3 times accept 5.

Input  : arr[] = {2, 2, 2, 10, 2}
            k = 4
Output : 10
Every element appears 4 times accept 10.
```

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one starting from the leftmost element. The inner loop checks if the element is present k times or not. If present, then ignores the element, else prints the element.

Time Complexity of above solution is $O(n^2)$. We can **Use Sorting** to solve the problem in O(nLogn) time. The idea is simple, first sort the array so that all occurrences of every element become consecutive. Once the occurrences become consecutive, we can traverse the sorted array and print the unique element in O(n) time.

We can **Use Hashing** to solve this in O(n) time on average. The idea is to traverse the given array from left to right and keep track of visited elements in a hash table. Finally print the element with count 1.

The hashing based solution requires O(n) extra space. We can **use bitwise AND** to find the unique element in O(n) time and constant extra space.

1. Create an array **count**[] of size equal to number of bits in binary representations of numbers.
2. Fill count array such that count[i] stores count of array elements with i-th bit set.
3. Form result using count array. We put 1 at a position i in result if count[i] is not multiple of k. Else we put 0.

```
 // CPP program to find unique element where
// every element appears k times except one
#include <bits/stdc++.h>
using namespace std;

int findUnique(unsigned int a[], int n, int k)
{
    // Create a count array to store count of
    // numbers that have a particular bit set.
    // count[i] stores count of array elements
    // with i-th bit set.
    int INT_SIZE = 8 * sizeof(unsigned int);
    int count[INT_SIZE];
    memset(count, 0, sizeof(count));

    // AND(bitwise) each element of the array
    // with each set digit (one at a time)
    // to get the count of set bits at each
    // position
    for (int i = 0; i < INT_SIZE; i++)
        for (int j = 0; j < n; j++)
            if ((a[j] & (1 << i)) != 0)
                count[i] += 1;

    // Now consider all bits whose count is
    // not multiple of k to form the required
    // number.
    unsigned res = 0;
    for (int i = 0; i < INT_SIZE; i++)
        res += (count[i] % k) * (1 << i);
    return res;
}

// Driver Code
int main()
{
    unsigned int a[] = { 6, 2, 5, 2, 2, 6, 6 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 3;
```

```
    cout << findUnique(a, n, k);
    return 0;
}
```

Output:

```
5
```

Source

https://www.geeksforgeeks.org/find-unique-element-element-occurs-k-times-except-one/

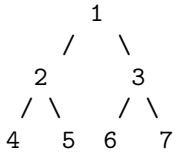
Chapter 294

Vertical Sum in a given Binary Tree | Set 1

Vertical Sum in a given Binary Tree | Set 1 - GeeksforGeeks

Given a Binary Tree, find vertical sum of the nodes that are in same vertical line. Print all sums through different vertical lines.

Examples:



The tree has 5 vertical lines

Vertical-Line-1 has only one node 4 => vertical sum is 4
Vertical-Line-2: has only one node 2=> vertical sum is 2
Vertical-Line-3: has three nodes: 1,5,6 => vertical sum is 1+5+6 = 12
Vertical-Line-4: has only one node 3 => vertical sum is 3
Vertical-Line-5: has only one node 7 => vertical sum is 7

So expected output is 4, 2, 12, 3 and 7

We need to check the Horizontal Distances from root for all nodes. If two nodes have the same Horizontal Distance (HD), then they are on same vertical line. The idea of HD is simple. HD for root is 0, a right edge (edge connecting to right subtree) is considered as +1 horizontal distance and a left edge is considered as -1 horizontal distance. For example, in the above tree, HD for Node 4 is at -2, HD for Node 2 is -1, HD for 5 and 6 is 0 and HD for node 7 is +2.
We can do inorder traversal of the given Binary Tree. While traversing the tree, we can

recursively calculate HDs. We initially pass the horizontal distance as 0 for root. For left subtree, we pass the Horizontal Distance as Horizontal distance of root minus 1. For right subtree, we pass the Horizontal Distance as Horizontal Distance of root plus 1.

Following is Java implementation for the same. HashMap is used to store the vertical sums for different horizontal distances. Thanks to Nages for suggesting this method.

Java

```
 import java.util.HashMap;

// Class for a tree node
class TreeNode {

    // data members
    private int key;
    private TreeNode left;
    private TreeNode right;

    // Accessor methods
    public int key()        { return key; }
    public TreeNode left()  { return left; }
    public TreeNode right() { return right; }

    // Constructor
    public TreeNode(int key)
   { this.key = key; left = null; right = null; }

    // Methods to set left and right subtrees
    public void setLeft(TreeNode left)   { this.left = left; }
    public void setRight(TreeNode right) { this.right = right; }
}

// Class for a Binary Tree
class Tree {

    private TreeNode root;

    // Constructors
    public Tree() { root = null; }
    public Tree(TreeNode n) { root = n; }

    // Method to be called by the consumer classes
    // like Main class
    public void VerticalSumMain() { VerticalSum(root); }

    // A wrapper over VerticalSumUtil()
    private void VerticalSum(TreeNode root) {
```

```
        // base case
        if (root == null) { return; }

        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM =
                   new HashMap<Integer, Integer>();

        // Calls the VerticalSumUtil() to store the
        // vertical sum values in hM
        VerticalSumUtil(root, 0, hM);

        // Prints the values stored by VerticalSumUtil()
        if (hM != null) {
            System.out.println(hM.entrySet());
        }
    }

    // Traverses the tree in Inoorder form and builds
    // a hashMap hM that contains the vertical sum
    private void VerticalSumUtil(TreeNode root, int hD,
                         HashMap<Integer, Integer> hM) {

        // base case
        if (root == null) {  return; }

        // Store the values in hM for left subtree
        VerticalSumUtil(root.left(), hD - 1, hM);

        // Update vertical sum for hD of this node
        int prevSum = (hM.get(hD) == null) ? 0 : hM.get(hD);
        hM.put(hD, prevSum + root.key());

        // Store the values in hM for right subtree
        VerticalSumUtil(root.right(), hD + 1, hM);
    }
}

// Driver class to test the verticalSum methods
public class Main {

    public static void main(String[] args) {
        /* Create following Binary Tree
              1
            /    \
          2        3
         / \      / \
        4   5    6   7
```

```
        */
        TreeNode root = new TreeNode(1);
        root.setLeft(new TreeNode(2));
        root.setRight(new TreeNode(3));
        root.left().setLeft(new TreeNode(4));
        root.left().setRight(new TreeNode(5));
        root.right().setLeft(new TreeNode(6));
        root.right().setRight(new TreeNode(7));
        Tree t = new Tree(root);

        System.out.println("Following are the values of" +
                           " vertical sums with the positions" +
                        " of the columns with respect to root ");
        t.VerticalSumMain();
    }
}
```

C++

```
 // C++ program to find Vertical Sum in
// a given Binary Tree
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    struct Node *left, *right;
};

// A utility function to create a new
// Binary Tree node
Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Traverses the tree in Inoorder form and
// populates a hashMap that contains the
// vertical sum
void verticalSumUtil(Node *node, int hd,
                     map<int, int> &Map)
{
    // Base case
```

```
    if (node == NULL) return;

    // Recur for left subtree
    verticalSumUtil(node->left, hd-1, Map);

    // Add val of current node to
    // map entry of corresponding hd
    Map[hd] += node->data;

    // Recur for right subtree
    verticalSumUtil(node->right, hd+1, Map);
}

// Function to find vertical sum
void verticalSum(Node *root)
{
    // a map to store sum of each horizontal
    // distance
    map < int, int> Map;
    map < int, int> :: iterator it;

    // populate the map
    verticalSumUtil(root, 0, Map);

    // Prints the values stored by VerticalSumUtil()
    for (it = Map.begin(); it != Map.end(); ++it)
    {
        cout << it->first << ": "
             << it->second << endl;
    }
}

// Driver program to test above functions
int main()
{
    // Create binary tree shown in above figure
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);

    cout << "Following are the values of vertical
            " sums with the positions of the "
```

```
            "columns with respect to root\n";
    verticalSum(root);

    return 0;
}
// This code is contributed by Aditi Sharma
```

Vertical Sum in Binary Tree | Set 2 (Space Optimized)

Time Complexity: O(n)

Source

https://www.geeksforgeeks.org/vertical-sum-in-a-given-binary-tree/

Chapter 295

Zoho On Campus Drive | Set 24 (Software Developer)

Zoho On Campus Drive | Set 24 (Software Developer) - GeeksforGeeks

Zoho had visited our college to recruit software developers. There were two types of salary package offered and also internships.
These were the rounds:

Round 1 – Online Test
There were 30 questions. 15 aptitude and 15 coding related (from C)

Aptitude was quite simple but the coding questions were really challenging.
The cut-off was pretty high as this was a round meant for mass filtering.

Round 2 – Coding

This was a hands-on coding round with 8 questions.
You could use C, C++ or JAVA

Most of the questions were easy:

Given 4 integers as input. Find whether they would form a rectangle, square or none.
Insert an element at a particular index in an array.

Some were average:

Given a large number convert it to the base 7.
Given an IP address validate it based on the given conditions.
Sort parts of an array separately using peak values.

Some were a bit challenging:

Given an input array, find the number of occurrences of a particular number without looping (use hashing)
Diamond pattern printing based on some conditions
Given an array of characters print the characters that have 'n' number of occurrences. If a character appears consecutively it is counted as 1 occurrence

Eg: **a** b **a a** b c c d e d
Here a has only 2 occurrences

Round 3 & 4 – Advanced coding:
In these rounds we were asked to implement the logic for 2 games
– Minesweeper
– Breakout a.k.a. Arkanoid a.k.a. Brick-Breaker (you'll find it online)

The game was split into various stages and we were asked to implement the logic stage by stage.

Interviews:

A few interviews followed. Some of which was just to get to know you. One interview had a few puzzles. And one was an official interview with a lead HR from Zoho

Those who did really well got the higher package and those who just missed it also got the job with a lower package.
Some were offered internships.
Overall it was a challenging process. Those with strong logical and coding skills would make it

Source

https://www.geeksforgeeks.org/zoho-campus-drive-set-24-software-developer/

Chapter 296

k-th distinct (or non-repeating) element in an array.

k-th distinct (or non-repeating) element in an array. - GeeksforGeeks

Given an integer array, print k-th distinct element in an array. The given array may contain duplicates and the output should print k-th element among all unique elements. If k is more than number of distinct elements, print -1.

Examples :

```
Input : arr[] = {1, 2, 1, 3, 4, 2},
        k = 2
Output : 4

First non-repeating element is 3
Second non-repeating element is 4

Input : arr[] = {1, 2, 50, 10, 20, 2},
        k = 3
Output : 10

Input : {2, 2, 2, 2},
        k = 2
Output : -1
```

A **simple solution** is to use two nested loops where outer loop picks elements from left to right, and inner loop checks if the picked element is present somewhere else. If not present, then increment count of distinct elements. If count becomes k, return current element.

C++

```
 // C++ program to print k-th distinct
// element in a given array
#include <bits/stdc++.h>
using namespace std;

// Returns k-th distinct
// element in arr.
int printKDistinct(int arr[], int n,
                               int k)
{
    int dist_count = 0;
    for (int i = 0; i < n; i++)
    {
        // Check if current element is
        // present somewhere else.
        int j;
        for (j = 0; j < n; j++)
            if (i != j && arr[j] == arr[i])
                break;

        // If element is unique
        if (j == n)
            dist_count++;

        if (dist_count == k)
            return arr[i];
    }

    return -1;
}

// Driver Code
int main ()
{
    int ar[] = {1, 2, 1, 3, 4, 2};
    int n = sizeof(ar) / sizeof(ar[0]);
    int k = 2;
    cout << printKDistinct(ar, n, k);
    return 0;
}
```

Java

```
 // Java program to print k-th distinct
// element in a given array
class GFG
{
    // Returns k-th distinct element in arr.
```

```
    static int printKDistinct(int arr[],
                              int n,
                              int k)
    {
        int dist_count = 0;
        for (int i = 0; i < n; i++)
        {

            // Check if current element is
            // present somewhere else.
            int j;

            for (j = 0; j < n; j++)
                if (i != j && arr[j] == arr[i])
                    break;

            // If element is unique
            if (j == n)
                dist_count++;

            if (dist_count == k)
                return arr[i];
        }

        return -1;
    }

    //Driver code
    public static void main (String[] args)
    {

        int ar[] = {1, 2, 1, 3, 4, 2};
        int n = ar.length;
        int k = 2;

        System.out.print(printKDistinct(ar, n, k));
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to print k-th distinct
// element in a given array
using System;

class GFG
```

```
{
    // Returns k-th distinct element in arr
    static int printKDistinct(int []arr,
                                int n,
                                int k)
    {

        int dist_count = 0;
        for (int i = 0; i < n; i++)
        {

            // Check if current element is
            // present somewhere else.
            int j;

            for (j = 0; j < n; j++)
                if (i != j && arr[j] == arr[i])
                    break;

            // If element is unique
            if (j == n)
                dist_count++;

            if (dist_count == k)
                return arr[i];
        }

        return -1;
    }

    //Driver code
    public static void Main ()
    {

        int []ar = {1, 2, 1, 3, 4, 2};
        int n = ar.Length;
        int k = 2;

        Console.Write(printKDistinct(ar, n, k));
    }
}

// This code is contributed by nitn mittal
```

PHP

```
 <?php
// PHP program to print k-th
```

```
// distinct element in a
// given array


// Returns k-th distinct
// element in arr.
function printKDistinct($arr,
                        $n, $k)
{
    $dist_count = 0;
    for ($i = 0; $i < $n; $i++)
    {
        // Check if current element
        // is present somewhere else.
        $j;
        for ($j = 0; $j < $n; $j++)
            if ($i != $j && $arr[$j] ==
                            $arr[$i])
                break;

        // If element is unique
        if ($j == $n)
            $dist_count++;

        if ($dist_count == $k)
            return $arr[$i];
    }

    return -1;
}

// Driver Code
$ar = array(1, 2, 1, 3, 4, 2);
$n = sizeof($ar) / sizeof($ar[0]);
$k = 2;
echo printKDistinct($ar, $n, $k);

// This code is contributed by nitin mittal.
?>
```

Output :

```
4
```

An **efficient solution** is to use Hashing to solve this in O(n) time on average.

1) Create an empty hash table.
2) Traverse input array from left to right and store elements and their counts in the hash table.
3) Traverse input array again from left to right. Keep counting elements with count as 1.
4) If count becomes k, return current element.

C++

```
 // C++ program to print k-th
// distinct element in a
// given array
#include <bits/stdc++.h>
using namespace std;

// Returns k-th distinct
// element in arr
int printKDistinct(int arr[],
                   int n, int k)
{
    // Traverse input array and
    // store counts if individual
    // elements.
    unordered_map<int, int> h;
    for (int i = 0; i < n; i++)
        h[arr[i]]++;

    // If size of hash is
    // less than k.
    if (h.size() < k)
        return -1;

    // Traverse array again and
    // find k-th element with
    // count as 1.
    int dist_count = 0;
    for (int i = 0; i < n; i++)
    {
        if (h[arr[i]] == 1)
            dist_count++;
        if (dist_count == k)
            return arr[i];
    }

    return -1;
}

// Driver Code
int main ()
```

```
{
    int ar[] = {1, 2, 1, 3, 4, 2};
    int n = sizeof(ar) / sizeof(ar[0]);
    cout << printKDistinct(ar, n, 2);
    return 0;
}
```

Python3

```
 # Python3 program to print k-th
# distinct element in a given array
def printKDistinct(arr, size, KthIndex):
    dict = {}
    vect = []
    for i in range(size):
        if(arr[i] in dict):
            dict[arr[i]] = dict[arr[i]] + 1
        else:
            dict[arr[i]] = 1
    for i in range(size):
        if(dict[arr[i]] > 1):
            continue
        else:
            KthIndex = KthIndex - 1
        if(KthIndex == 0):
            return arr[i]
    return -1

# Driver Code
arr = [1, 2, 1, 3, 4, 2]
size = len(arr)
print(printKDistinct(arr, size, 2))

# This code is contributed
# by Akhand Pratap Singh
```

Output :

```
4
```

Improved By : nitin mittal, Akhand Pratap Singh 3

Source

https://www.geeksforgeeks.org/k-th-distinct-or-non-repeating-element-in-an-array/

Chapter 297

k-th missing element in increasing sequence which is not present in a given sequence

k-th missing element in increasing sequence which is not present in a given sequence - GeeksforGeeks

Given two sequences, one is increasing sequence **a**[] and another a normal sequence **b**[], find the K-th missing element in the increasing sequence which is not present in the given sequence. If no k-th missing element is there output -1

Examples:

```
Input:  a[] = {0, 2, 4, 6, 8, 10, 12, 14, 15};
        b[] = {4, 10, 6, 8, 12};
          k = 3
Output: 14
Explanation : The numbers from increasing sequence that
are not present in the given sequence are 0, 2, 14, 15.
The 3rd missing number is 14.
```

n1 Number of elements on increasing sequence a[].
n2 Number of elements in given sequence b[].

A **naive approach** is to iterate for every element in the increasing sequence and check if it is present in the given sequence or not, and keep a counter of not present elements, and print the k-th non present element. This will not be efficient enough as it has two nested for loops which will take O(n2).
Time complexity: O(n1 * n2)
Auxiliary space: O(1)

An **efficient approach** is to use hashing. We store all elements of given sequence in a hash table. Then we iterate through all elements of increasing sequence. For every element, we search it in the hash table. If element is present in not hash table, then we increment count of missing elements. If count becomes k, we return the missing element.

Below is the C++ implementation of the above approach

```
 // C++ program to find the k-th missing element
// in a given sequence
#include <bits/stdc++.h>
using namespace std;

// Returns k-th missing element. It returns -1 if
// no k is more than number of missing elements.
int find(int a[], int b[], int k, int n1, int n2)
{
    // Insert all elements of givens sequence b[].
    unordered_set<int> s;
    for (int i = 0; i < n2; i++)
        s.insert(b[i]);

    // Traverse through increasing sequence and
    // keep track of count of missing numbers.
    int missing = 0;
    for (int i = 0; i < n1; i++) {
        if (s.find(a[i]) == s.end())
            missing++;
        if (missing == k)
            return a[i];
    }

    return -1;
}

// driver program to test the above function
int main()
{
    int a[] = { 0, 2, 4, 6, 8, 10, 12, 14, 15 };
    int b[] = { 4, 10, 6, 8, 12 };
    int n1 = sizeof(a) / sizeof(a[0]);
    int n2 = sizeof(b) / sizeof(b[0]);

    int k = 3;
    cout << find(a, b, k, n1, n2);
    return 0;
}
```

Output:

```
14
```

Time complexity: O(n1 + n2)
Auxiliary Space: O(n2)

Source

https://www.geeksforgeeks.org/k-th-missing-element-increasing-sequence-not-present-given-sequence/

Chapter 298

set vs unordered_set in C++ STL

set vs unordered_set in C++ STL - GeeksforGeeks

Pre-requisite : set in C++, unordered_set in C++

Differences :

```
                |     set              | unordered_set
-------------------------------------------------------
Ordering        | increasing  order    | no ordering
                | (by default)         |

Implementation  | Self balancing BST   | Hash Table
                | like Red-Black Tree  |

search time     | log(n)               | O(1) -> Average
                |                      | O(n) -> Worst Case

Insertion time  | log(n) + Rebalance   | Same as search

Deletion time   | log(n) + Rebalance   | Same as search
```

Use set when

- We need ordered data.
- We would have to print/access the data (in sorted order).
- We need predecessor/successor of elements.
- Since set is ordered, we can use functions like binary_search(), lower_bound() and upper_bound() on set elements. These functions cannot be used on unordered_set().
- See advantages of BST over Hash Table for more cases.

Use unordered_set when

- We need to keep a set of distinct elements and no ordering is required.
- We need single element access i.e. no traversal.

Examples:

```
set:
Input  :  1, 8, 2, 5, 3, 9
Output :  1, 2, 3, 5, 8, 9

Unordered_set:
Input  : 1, 8, 2, 5, 3, 9
Output : 9 3 1 8 2 5
```

If you want to look at implementation details of set and unordered_set in c++ STL, see Set Vs Map. Set allows to traverse elements in sorted order whereas Unordered_set doesn't allow to traverse elements in sorted order.

```
 // Program to print elements of set
#include <bits/stdc++.h>
using namespace std;

int main()
{
    set<int> s;
    s.insert(5);
    s.insert(1);
    s.insert(6);
    s.insert(3);
    s.insert(7);
    s.insert(2);

    cout << "Elements of set in sorted order: \n";
    for (auto it : s)
        cout << it << " ";

    return 0;
}
```

Output:

```
Elements of set in sorted order:
1 2 3 5 6 7
```

```
 // Program to print elements of set
#include <bits/stdc++.h>
using namespace std;

int main()
{
    unordered_set<int> s;
    s.insert(5);
    s.insert(1);
    s.insert(6);
    s.insert(3);
    s.insert(7);
    s.insert(2);

    cout << "Elements of unordered_set: \n";
    for (auto it : s)
        cout << it << " ";

    return 0;
}
```

Output:

```
Elements of unordered_set:
2 7 5 1 6 3
```

Predecessor/Successor in Set:
Set can be modified to find predecessor or successor whereas Unordered_set doesn't allow to find predecessor/Successor.

```
 // Program to print inorder predecessor and inorder successor
#include <bits/stdc++.h>
using namespace std;

set<int> s;

void inorderPredecessor(int key)
{
    if (s.find(key) == s.end()) {
        cout << "Key doesn't exist\n";
        return;
    }

    set<int>::iterator it;
    it = s.find(key); // get iterator of key
```

```
    // If iterator is at first position
    // Then, it doesn't have predecessor
    if (it == s.begin()) {
        cout << "No predecessor\n";
        return;
    }

    --it; // get previous element
    cout << "predecessor of " << key << " is=";
    cout << *(it) << "\n";
}

void inorderSuccessor(int key)
{
    if (s.find(key) == s.end()) {
        cout << "Key doesn't exist\n";
        return;
    }

    set<int>::iterator it;
    it = s.find(key); // get iterator of key
    ++it; // get next element

    // Iterator points to NULL (Element does
    // not exist)
    if (it == s.end())
    {
        cout << "No successor\n";
        return;
    }
    cout << "successor of " << key << " is=";
    cout << *(it) << "\n";
}

int main()
{
    s.insert(1);
    s.insert(5);
    s.insert(2);
    s.insert(9);
    s.insert(8);

    inorderPredecessor(5);
    inorderPredecessor(1);
    inorderPredecessor(8);
    inorderSuccessor(5);
    inorderSuccessor(2);
    inorderSuccessor(9);
```

```
    return 0;
}
```

Output:

```
predecessor of 5 is=2
No predecessor
predecessor of 8 is=5
successor of 5 is=8
successor of 2 is=5
No successor
```

Source

https://www.geeksforgeeks.org/set-vs-unordered_set-c-stl/