

HANDBOOK OF ALGORITHMS

Courtesy of
GeeksforGeeks

Fahim Ahmed

Bangladesh

HANDBOOK OF ALGORITHMS.pdf

Search and Sort Cover.pdf

Searching and Sorting.pdf

Greedy Algorithms.pdf

Greedy.pdf

Divide cover.pdf

Divide and Conquer.pdf

backtrack cover.pdf

Backtracking.pdf

Branch and Bound Cover.pdf

Branch and Bound.pdf

Mathematics cover.pdf

Mathematics.pdf

Geometry Cover.pdf

Geometry Algorithms.pdf

DP cover.pdf

Dynamic Programming.pdf

pattern cover.pdf

String and Pattern Searching.pdf

HANDBOOK OF ALGORITHMS

Section
Searching & Sorting

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Linear Search

Problem: Given an array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do **linear search**, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.

Example:

An array with 10 elements, search for “9”:

56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9
56	3	249	518	7	26	94	651	23	9

```
// Linearly search x in arr[]. If x is present then return its
// location, otherwise return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i=0; i<n; i++)

```

```

    if (arr[i] == x)
        return i;
    return -1;
}

```

[Run on IDE](#)

Python

```

# Searching an element in a list/array in python
# can be simply done using 'in' operator
# Example:
# if x in arr:
#     print arr.index(x)

# If you want to implement Linear Search in python

# Linearly search x in arr[]
# If x is present then return its location
# else return -1

def search(arr, x):

    for i in range(len(arr)):

        if arr[i] == x:
            return i

    return -1

```

[Run on IDE](#)

Java

```

class LinearSearch
{
    // This function returns index of element x in arr[]
    static int search(int arr[], int n, int x)
    {
        for (int i = 0; i < n; i++)
        {
            // Return the index of the element if the element
            // is found
            if (arr[i] == x)
                return i;
        }

        // return -1 if the element is not found
        return -1;
    }
}

```

[Run on IDE](#)

The time complexity of above algorithm is $O(n)$.

Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to Linear search.

Also See – [Binary Search](#)

Image Source : http://www.nyckidd.com/bob/Linear%20Search%20and%20Binary%20Search_WorkingCopy.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner

The advertisement features a dark red background. At the top, there's a small blue info icon and a light blue close icon. Below them, the title "Introduction to Computer Science" is displayed in large white font. Underneath the title, the text "ENROLLMENT IS OPEN TO EVERYONE" is shown in smaller white font. A large white button with a dark red border contains the text "Start Now" in dark red. At the bottom, the edX logo (a stylized "edX" with "www.edx.org" below it) is positioned next to the Harvard University logo (a crest with "HARVARD" and "UNIVERSITY" text).

See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Searching and Sorting

([Login](#) to Rate and Mark)

Average Rating : 3.3/5.0

Average Difficulty : 1.1/5.0

1.1

Based on 36 vote(s)



Add to TODO List

Mark as DONE

[Load Comments](#)

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Binary Search

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do [linear search](#). The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Example:

If searching for 23 in the 10-element array:

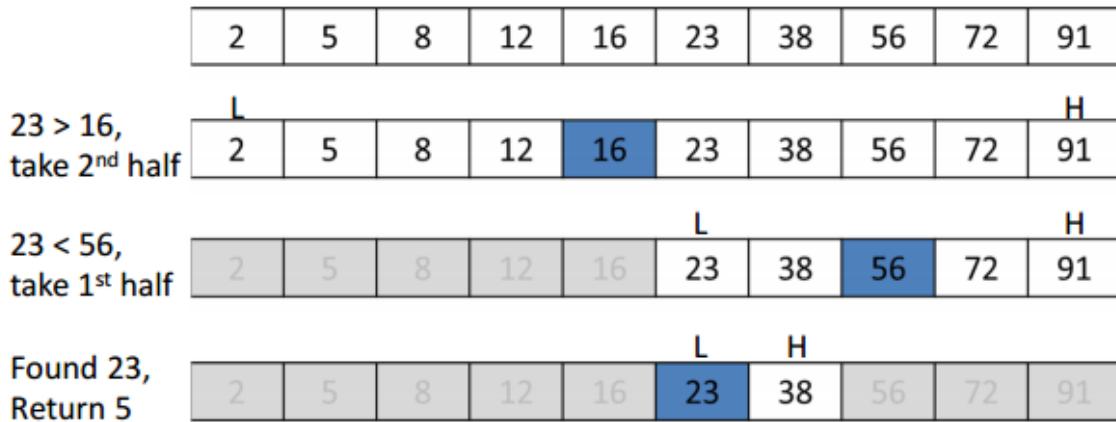


Image Source : http://www.nyckidd.com/bob/Linear%20Search%20and%20Binary%20Search_WorkingCopy.pdf

The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Logn).

We strongly recommend that you click here and practice it, before moving on to the solution.

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.

3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Recursive implementation of Binary Search

```
#include <stdio.h>

// A recursive binary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - 1)/2;

        // If the element is present at the middle itself
        if (arr[mid] == x) return mid;

        // If element is smaller than mid, then it can only be present
        // in left subarray
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present in right subarray
        return binarySearch(arr, mid+1, r, x);
    }

    // We reach here when element is not present in array
    return -1;
}

int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
                  : printf("Element is present at index %d", result);
    return 0;
}
```

[Run on IDE](#)

Python

```
# Python Program for recursive binary search.

# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - 1)/2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it can only
        # be present in left subarray
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)
```

```

# Else the element can only be present in right subarray
else:
    return binarySearch(arr, mid+1, r, x)

else:
    # Element is not present in the array
    return -1

# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"

```

[Run on IDE](#)

Java

```

// Java implementation of recursive Binary Search
class BinarySearch
{
    // Returns index of x if it is present in arr[l..r], else
    // return -1
    int binarySearch(int arr[], int l, int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - l)/2;

            // If the element is present at the middle itself
            if (arr[mid] == x)
                return mid;

            // If element is smaller than mid, then it can only
            // be present in left subarray
            if (arr[mid] > x)
                return binarySearch(arr, l, mid-1, x);

            // Else the element can only be present in right
            // subarray
            return binarySearch(arr, mid+1, r, x);
        }

        // We reach here when element is not present in array
        return -1;
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = {2,3,4,10,40};
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr,0,n-1,x);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index "+result);
    }
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

```
Element is present at index 3
```

Iterative implementation of Binary Search

```
#include <stdio.h>

// A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r-1)/2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was not present
    return -1;
}

int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
                  : printf("Element is present at index %d", result);
    return 0;
}
```

[Run on IDE](#)

Python

```
# Iterative Binary Search Function
# It returns location of x in given array arr if present,
# else returns -1
def binarySearch(arr, l, r, x):

    while l <= r:

        mid = l + (r - 1)/2;

        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1
```

```

# If x is smaller, ignore right half
else:
    r = mid - 1

# If we reach here, then the element was not present
return -1

# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"

```

[Run on IDE](#)

Java

```

// Java implementation of iterative Binary Search
class BinarySearch
{
    // Returns index of x if it is present in arr[], else
    // return -1
    int binarySearch(int arr[], int x)
    {
        int l = 0, r = arr.length - 1;
        while (l <= r)
        {
            int m = l + (r-l)/2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
        }

        // if we reach here, then element was not present
        return -1;
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = {2, 3, 4, 10, 40};
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr, x);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index "+result);
    }
}

```

[Run on IDE](#)

Output:

```
Element is present at index 3
```

Time Complexity:

The time complexity of Binary Search can be written as

```
T(n) = T(n/2) + c
```

The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is $\Theta(\log n)$.

Auxiliary Space: $O(1)$ in case of iterative implementation. In case of recursive implementation, $O(\log n)$ recursion call stack space.

Algorithmic Paradigm: Divide and Conquer



Interesting articles based on Binary Search.

- [The Ubiquitous Binary Search](#)
- [Interpolation search vs Binary search](#)
- [Find the minimum element in a sorted and rotated array](#)
- [Find a peak element](#)
- [Find a Fixed Point in a given array](#)
- [Count the number of occurrences in a sorted array](#)
- [Median of two sorted arrays](#)
- [Floor and Ceiling in a sorted array](#)
- [Find the maximum element in an array which is first increasing and then decreasing](#)

Coding Practice Questions on Binary Search

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner

The advertisement features a dark red header with white text. At the top right are icons for help and close. Below the header, the title "Introduction to Computer Science" is displayed in large white font. Underneath the title, the text "ENROLLMENT IS OPEN TO EVERYONE" is shown in smaller white font. A large white button in the center contains the text "Start Now" in black. At the bottom, there are logos for "edX" and "HARVARD UNIVERSITY".

See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Searching and Sorting

([Login](#) to Rate and Mark)

Average Rating : **3.5/5.0**

Based on **8** vote(s)



1.4

Average Difficulty : **1.4/5.0**

Based on **83** vote(s)



Add to TODO List

Mark as DONE

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Jump Search

Like [Binary Search](#), Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than [linear search](#)) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.

For example, suppose we have an array arr[] of size n and block (to be jumped) size m. Then we search at the indexes arr[0], arr[m], arr[2m].....arr[km] and so on. Once we find the interval ($\text{arr}[km] < x < \text{arr}[(k+1)m]$), we perform a linear search operation from the index km to find the element x.

Let's consider the following array: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610). Length of the array is 16. Jump search will find the value of 55 with the following steps assuming that the block size to be jumped is 4.

STEP 1: Jump from index 0 to index 4;

STEP 2: Jump from index 4 to index 8;

STEP 3: Jump from index 8 to index 16;

STEP 4: Since the element at index 16 is greater than 55 we will jump back a step to come to index 9.

STEP 5: Perform linear search from index 9 to get the element 55.

What is the optimal block size to be skipped?

In the worst case, we have to do n/m jumps and if the last checked value is greater than the element to be searched for, we perform $m-1$ comparisons more for linear search. Therefore the total number of comparisons in the worst case will be $((n/m) + m-1)$. The value of the function $((n/m) + m-1)$ will be minimum when $m = \sqrt{n}$. Therefore, the best step size is $m = \sqrt{n}$.

Java Implementation :

```
// Java program to implement Jump Search.
public class JumpSearch
{
    public static int jumpSearch(int[] arr, int x)
    {
        int n = arr.length;

        // Finding block size to be jumped
        int step = (int)Math.floor(Math.sqrt(n));

        // Finding the block where element is
        // present (if it is present)
        int prev = 0;
        while (arr[Math.min(step, n)-1] < x)
        {
            prev = step;
            step += (int)Math.floor(Math.sqrt(n));
            if (prev >= n)
                return -1;
        }
    }
}
```

```

// Doing a linear search for x in block
// beginning with prev.
while (arr[prev] < x)
{
    prev++;

    // If we reached next block or end of
    // array, element is not present.
    if (prev == Math.min(step, n))
        return -1;
}

// If element is found
if (arr[prev] == x)
    return prev;

return -1;
}

// Driver program to test function
public static void main(String [ ] args)
{
    int arr[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21,
                 34, 55, 89, 144, 233, 377, 610};
    int x = 55;

    // Find the index of 'x' using Jump Search
    int index = jumpSearch(arr, x);

    // Print the index where 'x' is located
    System.out.println("\nNumber " + x +
                       " is at index " + index);
}
}

```

[Run on IDE](#)

Output:

Number 55 is at index 10

Time Complexity : $O(\sqrt{n})$

Auxiliary Space : $O(1)$

Important points:

- Works only sorted arrays.
- The optimal size of a block to be jumped is $O(\sqrt{n})$. This makes the time complexity of Jump Search $O(\sqrt{n})$.
- The time complexity of Jump Search is between Linear Search ($O(n)$) and Binary Search ($O(\log n)$).
- Binary Search is better than Jump Search, but Jump search has an advantage that we traverse back only once (Binary Search may require up to $O(\log n)$ jumps, consider a situation where the element to be search is the smallest element or smaller than the smallest). So in a systems where jumping back is costly, we use Jump Search.

References:

https://en.wikipedia.org/wiki/Jump_search

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

Searching

Related Posts:

- Second minimum element using minimum comparisons
- [Interpolation Search](#)
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays
- Search an element in an array where difference between adjacent elements is 1

(Login to Rate and Mark)

1.8

Average Difficulty : 1.8/5.0
Based on 18 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Interpolation Search

Given a sorted array of n uniformly distributed values arr[], write a function to search for a particular element x in the array.

Linear Search finds the element in O(n) time, [Jump Search](#) takes O(\sqrt{n}) time and [Binary Search](#) take O(Log n) time.

The Interpolation Search is an improvement over [Binary Search](#) for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to middle element to check. On the other hand interpolation search may go to different locations according the value of key being searched. For example if the value of key is closer to the last element, interpolation search is likely to start search toward the end side.

To find the position to be searched, it uses following formula.

```
// The idea of formula is to return higher value of pos
// when element to be searched is closer to arr[hi]. And
// smaller value when closer to arr[lo]
pos = lo + [ (x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo]) ]

arr[] ==> Array where elements need to be searched
x      ==> Element to be searched
lo     ==> Starting index in arr[]
hi     ==> Ending index in arr[]
```

Algorithm

Rest of the Interpolation algorithm is same except the above partition logic.

Step1: In a loop, calculate the value of “pos” using the probe position formula.

Step2: If it is a match, return the index of the item, and exit.

Step3: If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise calculate the same in the right sub-array.

Step4: Repeat until a match is found or the sub-array reduces to zero.

Below is C implementation of algorithm.

```
// C program to implement interpolation search
#include<stdio.h>

// If x is present in arr[0..n-1], then returns
// index of it, else returns -1.
int interpolationSearch(int arr[], int n, int x)
{
    // Find indexes of two corners
    int lo = 0, hi = (n - 1);
```

```

// Since array is sorted, an element present
// in array must be in range defined by corner
while (lo <= hi && x >= arr[lo] && x <= arr[hi])
{
    // Probing the position with keeping
    // uniform distribution in mind.
    int pos = lo + (((double)(hi-lo) /
        (arr[hi]-arr[lo]))*(x - arr[lo]));

    // Condition of target found
    if (arr[pos] == x)
        return pos;

    // If x is larger, x is in upper part
    if (arr[pos] < x)
        lo = pos + 1;

    // If x is smaller, x is in lower part
    else
        hi = pos - 1;
}
return -1;
}

// Driver Code
int main()
{
    // Array of items on which search will
    // be conducted.
    int arr[] = {10, 12, 13, 16, 18, 19, 20, 21, 22, 23,
                24, 33, 35, 42, 47};
    int n = sizeof(arr)/sizeof(arr[0]);

    int x = 11; // Element to be searched
    int index = interpolationSearch(arr, n, x);

    // If element was found
    if (index != -1)
        printf("Element found at index %d", index);
    else
        printf("Element not found.");
    return 0;
}

```

[Run on IDE](#)

Output :

Element found at index 4

Time Complexity : If elements are uniformly distributed, then **O (log log n)**. In worst case it can take upto O(n).

Auxiliary Space : O(1)

This article is contributed by **Aayu sachdev**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Searching Technical Scripter

Related Posts:

- Second minimum element using minimum comparisons
- Jump Search
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays
- Search an element in an array where difference between adjacent elements is 1

(Login to Rate and Mark)

3

Average Difficulty : 3/5.0
Based on 13 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksQuiz

Computer Science Quizzes for Geeks !

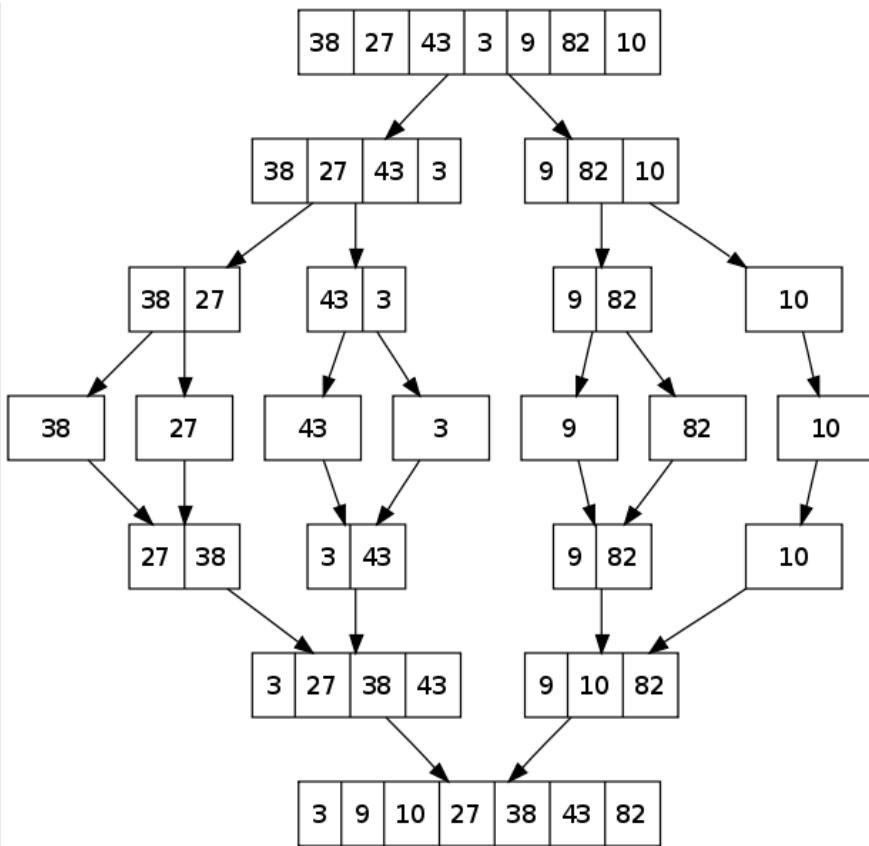
GATE CS Coding Practice Placements GeeksforGeeks

Merge Sort

Like [QuickSort](#), Merge Sort is a **Divide and Conquer** algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The `merge(arr, l, m, r)` is key process that assumes that `arr[l..m]` and `arr[m+1..r]` are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

```
MergeSort(arr[], l, r)
If r > 1
    1. Find the middle point to divide the array into two halves:
        middle m = (l+r)/2
    2. Call mergeSort for first half:
        Call mergeSort(arr, l, m)
    3. Call mergeSort for second half:
        Call mergeSort(arr, m+1, r)
    4. Merge the two halves sorted in step 2 and 3:
        Call merge(arr, l, m, r)
```

The following diagram from [wikipedia](#) shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.



We strongly recommend that you [click here](#) and practice it, before moving on to the solution.

```

/* C program for Merge Sort */
#include<stdlib.h>
#include<stdio.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy remaining elements of L[], if any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy remaining elements of R[], if any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
  
```

```

        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy the remaining elements of L[], if there
   are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there
   are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-1)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}

```

[Run on IDE](#)

Java

```
/* Java program for Merge Sort */
class MergeSort
{
    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    void merge(int arr[], int l, int m, int r)
    {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        /* Create temp arrays */
        int L[] = new int [n1];
        int R[] = new int [n2];

        /*Copy data to temp arrays*/
        for (int i=0; i<n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            R[j] = arr[m + 1+ j];

        /* Merge the temp arrays */

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarry array
        int k = l;
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
            }
            else
            {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        /* Copy remaining elements of L[] if any */
        while (i < n1)
        {
            arr[k] = L[i];
            i++;
            k++;
        }

        /* Copy remaining elements of R[] if any */
        while (j < n2)
        {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    // Main function that sorts arr[l..r] using
    // merge()
    void sort(int arr[], int l, int r)
    {
        if (l < r)
        {
            // Find the middle point
            int m = l + (r - l) / 2;
```

```

        int m = (l+r)/2;

        // Sort first and second halves
        sort(arr, l, m);
        sort(arr , m+1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver method
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};

    System.out.println("Given Array");
    printArray(arr);

    MergeSort ob = new MergeSort();
    ob.sort(arr, 0, arr.length-1);

    System.out.println("\nSorted array");
    printArray(arr);
}
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# Python program for implementation of MergeSort

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # Create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    # Merge the temp arrays back into arr[l..r]
    i = 0      # Initial index of first subarray
    j = 0      # Initial index of second subarray
    k = l      # Initial index of merged subarray

    while i < n1 and j < n2 :
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:

```

```

        arr[k] = R[j]
        j += 1
        k += 1

    # Copy the remaining elements of L[], if there
    # are any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if there
    # are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr,l,r):
    if l < r:

        # Same as (l+r)/2, but avoids overflow for
        # large l and h
        m = (l+(r-1))/2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)

# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print ("Given array is")
for i in range(n):
    print ("%d" %arr[i]),

mergeSort(arr,0,n-1)
print ("\n\nSorted array is")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra

```

Run on IDE

Output:

```
Given array is
12 11 13 5 6 7
```

```
Sorted array is
5 6 7 11 12 13
```

Time Complexity: Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$$T(n) = 2T(n/2) + \Theta(n)$$

The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is $\Theta(n \log n)$.

Time complexity of Merge Sort is $\Theta(n \log n)$ in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.

Auxiliary Space: O(n)

Algorithmic Paradigm: Divide and Conquer

Sorting In Place: No in a typical implementation

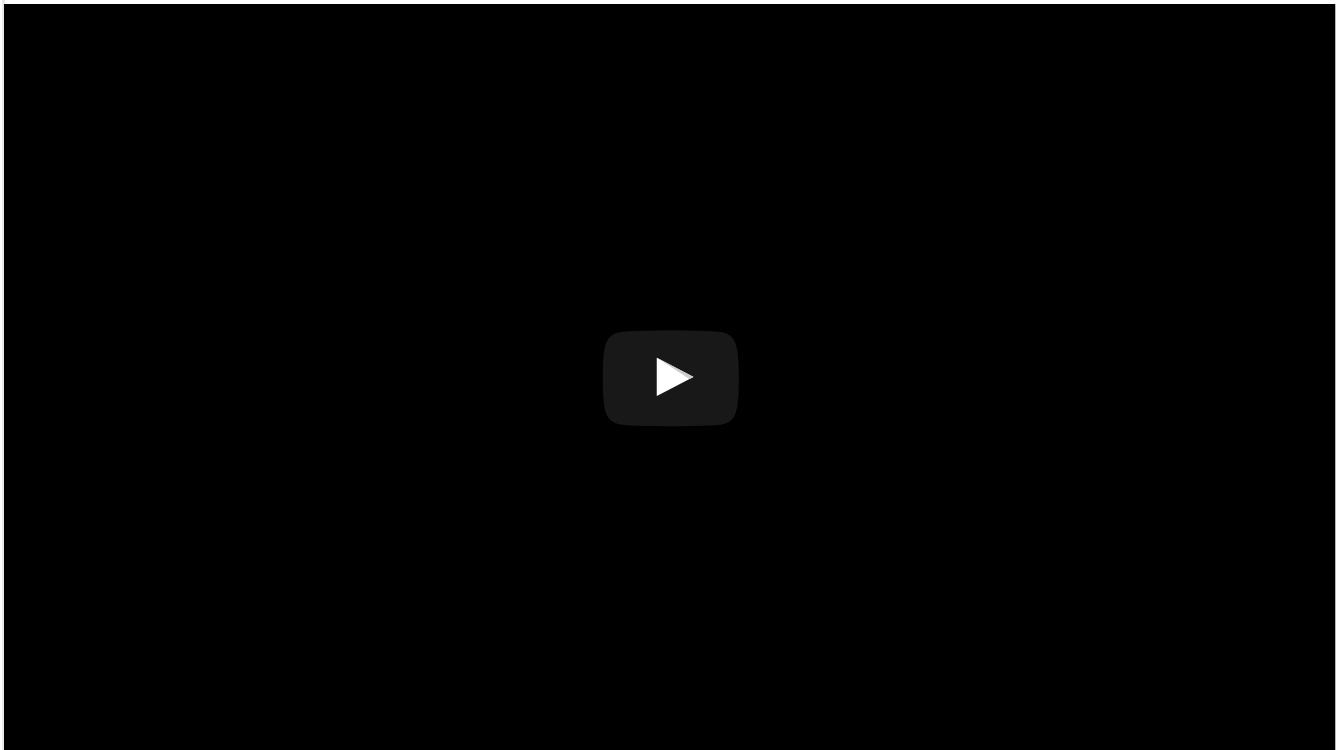
Stable: Yes

Applications of Merge Sort

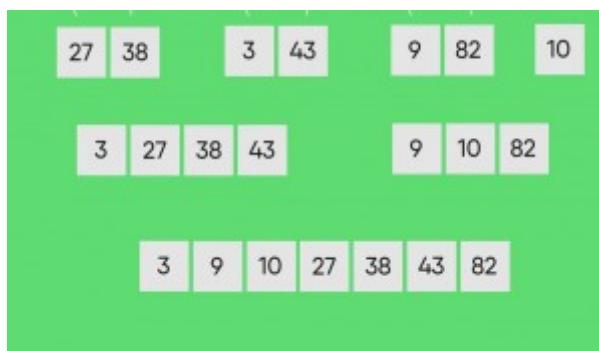
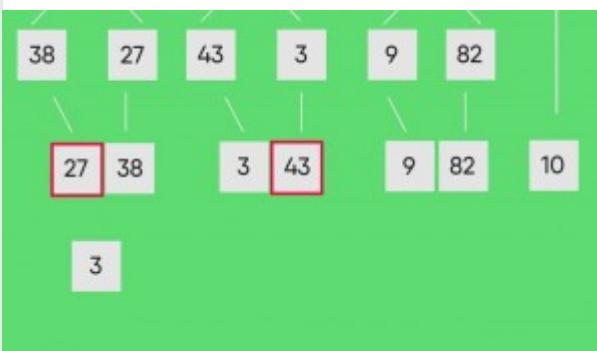
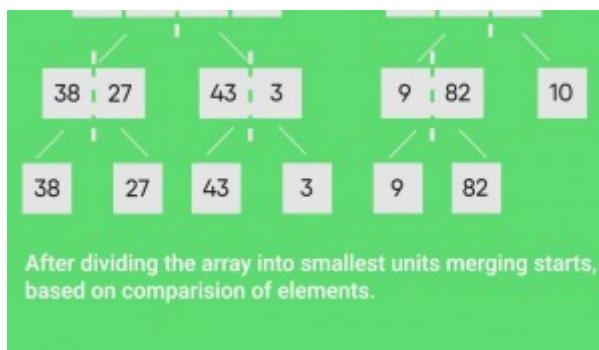
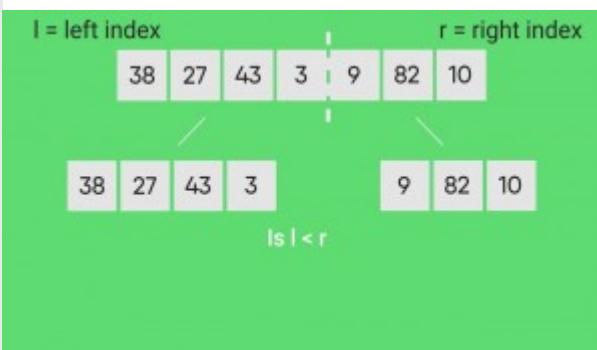
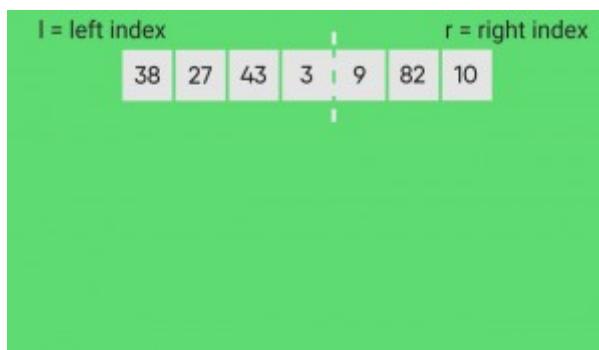
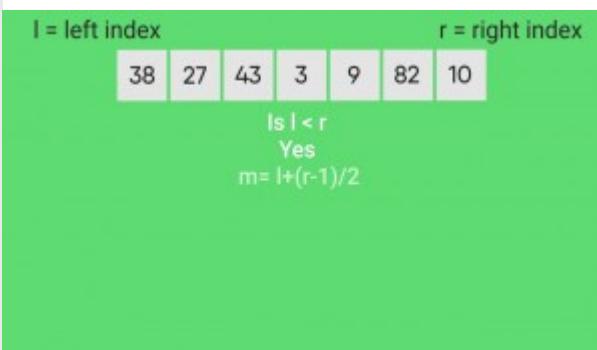
1. Merge Sort is useful for sorting linked lists in O($n\log n$) time. In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the memory at $(x + i \cdot 4)$. Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

2. Inversion Count Problem
3. Used in External Sorting



Snapshots:



Quiz on Merge Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort

Coding practice for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner



Download

Zip, Unzip or Open Any File
With Unzipper.

Unzipper



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Searching and Sorting

([Login](#) to Rate and Mark)

Average Rating : **4.6/5.0**

Average Difficulty : **2.8/5.0**

2.8

Based on **58** vote(s)

Based on **8** vote(s)



[Add to TODO List](#)

[Mark as DONE](#)

[Load Comments](#)

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Heap Sort

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

What is Binary Heap?

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible (Source [Wikipedia](#))

A **Binary Heap** is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Why array based representation for Binary Heap?

Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index i , the left child can be calculated by $2 * i + 1$ and right child by $2 * i + 2$ (assuming the indexing starts at 0).

Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

Lets understand with the help of an example:

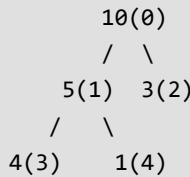
```
Input data: 4, 10, 3, 5, 1
          4(0)
           /   \
         10(1)  3(2)
          /   \
        5(3)  1(4)
```

The numbers in bracket represent the indices in the array representation of data.

Applying heapify procedure to index 1:

```
          4(0)
           /   \
         10(1)  3(2)
          /   \
        5(3)  1(4)
```

Applying heapify procedure to index 0:



The heapify procedure calls itself recursively to build heap in top down manner.

```

// C++ program for implementation of Heap Sort
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2*i + 1; // left = 2*i + 1
    int r = 2*i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program
int main()

```

```

{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}

```

[Run on IDE](#)

Java

```

// Java program for implementation of Heap Sort
public class HeapSort
{
    public void sort(int arr[])
    {
        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i=n-1; i>=0; i--)
        {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int n, int i)
    {
        int largest = i; // Initialize largest as root
        int l = 2*i + 1; // left = 2*i + 1
        int r = 2*i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

        // If largest is not root
        if (largest != i)
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
    }
}

```

```

        System.out.print(arr[i]+" ");
        System.out.println();
    }

// Driver program
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = arr.length;

    HeapSort ob = new HeapSort();
    ob.sort(arr);

    System.out.println("Sorted array is");
    printArray(arr);
}
}

```

[Run on IDE](#)

Python

```

# Python program for implementation of heap Sort

# To heapify subtree rooted at index i.
# n is size of heap
def heapify(arr, n, i):
    largest = i # Initialize largest as root
    l = 2 * i + 1      # left = 2*i + 1
    r = 2 * i + 2      # right = 2*i + 2

    # See if left child of root exists and is
    # greater than root
    if l < n and arr[i] < arr[l]:
        largest = l

    # See if right child of root exists and is
    # greater than root
    if r < n and arr[largest] < arr[r]:
        largest = r

    # Change root, if needed
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i] # swap

    # Heapify the root.
    heapify(arr, n, largest)

# The main function to sort an array of given size
def heapSort(arr):
    n = len(arr)

    # Build a maxheap.
    for i in range(n, -1, -1):
        heapify(arr, n, i)

    # One by one extract elements
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i] # swap
        heapify(arr, i, 0)

# Driver code to test above
arr = [ 12, 11, 13, 5, 6, 7]
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
    print ("%d" %arr[i]),
# This code is contributed by Mohit Kumra

```

[Run on IDE](#)

Output:

```
Sorted array is
5 6 7 11 12 13
```

[Here](#) is previous C code for reference.

Notes:

Heap sort is an in-place algorithm.

Its typical implementation is not stable, but can be made stable ([See this](#))

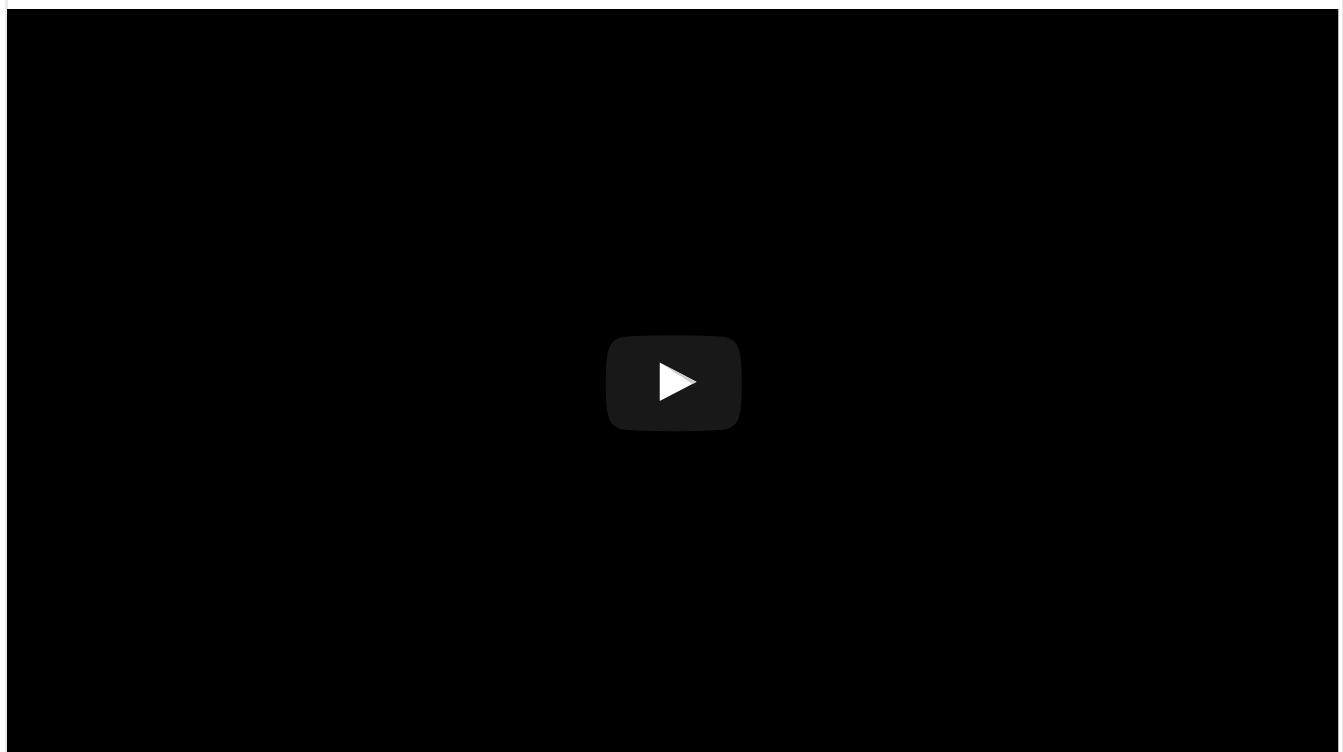
Time Complexity: Time complexity of heapify is $O(\log n)$. Time complexity of createAndBuildHeap() is $O(n)$ and overall time complexity of Heap Sort is $O(n \log n)$.

Applications of HeapSort

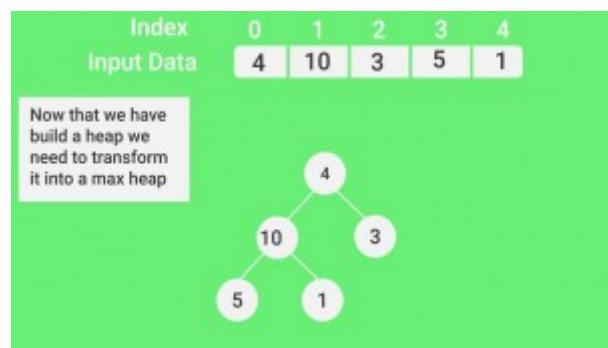
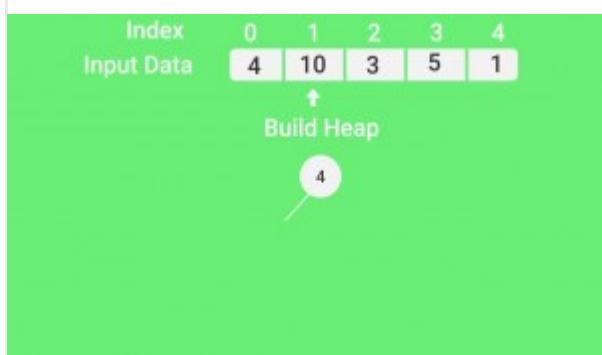
1. Sort a nearly sorted (or K sorted) array

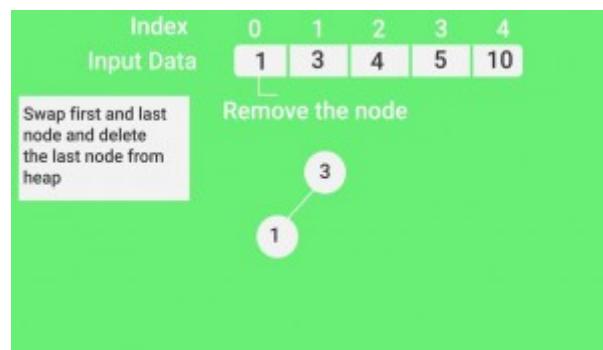
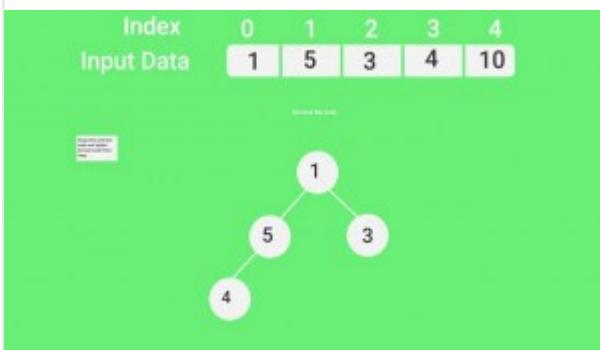
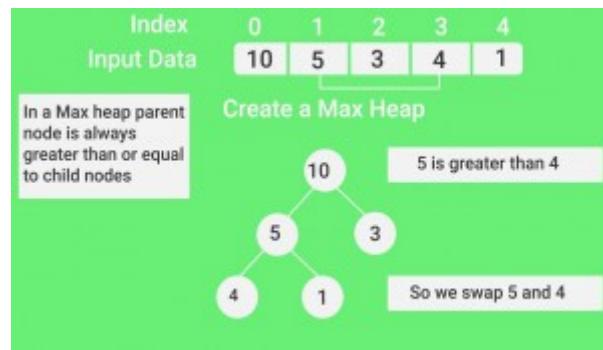
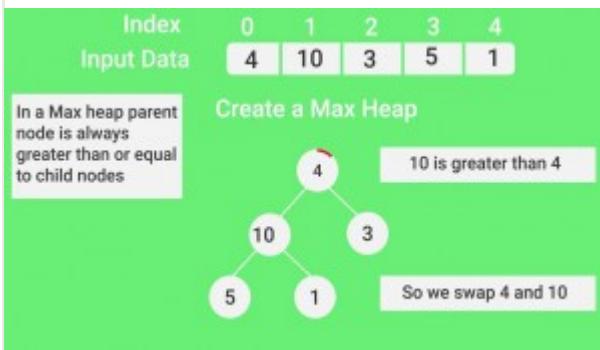
2. k largest(or smallest) elements in an array

Heap sort algorithm has limited uses because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used. See [Applications of Heap Data Structure](#)



Snapshots:





Quiz on Heap Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

QuickSort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort, Pigeonhole Sort

Coding practice for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner



Free Download

Zip, Unzip or Open Any File.

Unzipper



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Searching and Sorting

([Login](#) to Rate and Mark)

Average Rating : **4.8/5.0**

Average Difficulty : **2.9/5.0**

2.9

Based on **66** vote(s)

Based on **10** vote(s)



[Add to TODO List](#)

[Mark as DONE](#)

[Load Comments](#)

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

QuickSort

Like [Merge Sort](#), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

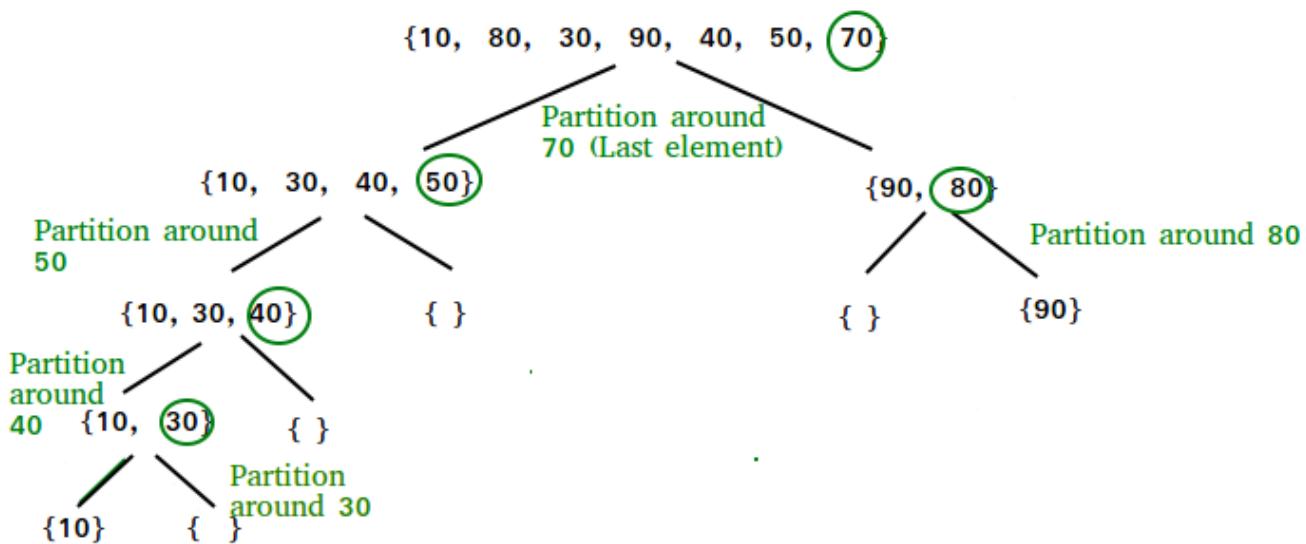
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```



Partition Algorithm

There can be many ways to do partition, following pseudo code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Pseudo code for partition()

```
/* 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 */
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];

    i = (low - 1) // Index of smaller element

    for (j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++; // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}
```

Illustration of partition() :

```
arr[] = {10, 80, 30, 90, 40, 50, 70}
Indexes: 0 1 2 3 4 5 6
```

```
low = 0, high = 6, pivot = arr[h] = 70
Initialize index of smaller element, i = -1
```

```
Traverse elements from j = low to high-1
j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 0
```

```

arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j
                                         // are same

j = 1 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 1
arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 2
arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped
j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]
i = 3
arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

```

We come out of loop because j is now equal to high-1.

Finally we place pivot at correct position by swapping

arr[i+1] and arr[high] (or pivot)

arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

We strongly recommend that you click here and practice it, before moving on to the solution.

Implementation:

Following are C++, Java and Python implementations of QuickSort.

```

/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* 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 */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];        // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {

```

```

        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;      // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program for implementation of QuickSort
class 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 */
    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-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 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 */
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);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i]+" ");
    System.out.println();
}

// Driver program
public static void main(String args[])
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = arr.length;

    QuickSort ob = new QuickSort();
    ob.sort(arr, 0, n-1);

    System.out.println("sorted array");
    printArray(arr);
}
/*This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# Python program for implementation of Quicksort Sort

# 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
def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return i + 1

```

```

# 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
def partition(arr,low,high):
    i = ( low-1 )           # index of smaller element
    pivot = arr[high]        # pivot

    for j in range(low , high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra

```

[Run on IDE](#)

Output:

```

Sorted array:
1 5 7 8 9 10

```

Analysis of QuickSort

Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + \theta(n)$$

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.

The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

$$T(n) = T(0) + T(n-1) + \theta(n)$$

which is equivalent to

$$T(n) = T(n-1) + \theta(n)$$

The solution of above recurrence is $\theta(n^2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

$$T(n) = 2T(n/2) + \theta(n)$$

The solution of above recurrence is $\theta(n\log n)$. It can be solved using case 2 of [Master Theorem](#).

Average Case:

To do average case analysis, we need to [consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy](#).

We can get an idea of average case by considering the case when partition puts $O(n/9)$ elements in one set and $O(9n/10)$ elements in other set. Following is recurrence for this case.

$$T(n) = T(n/9) + T(9n/10) + \theta(n)$$

Solution of above recurrence is also $O(n\log n)$

Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like [Merge Sort](#) and [Heap Sort](#), QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

What is 3-Way QuickSort?

In simple QuickSort algorithm, we select an element as pivot, partition the array around pivot and recur for subarrays on left and right of pivot.

Consider an array which has many redundant elements. For example, {1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 2, 2, 4, 1, 4, 4, 4}. If 4 is picked as pivot in Simple QuickSort, we fix only one 4 and recursively process remaining occurrences.

In 3 Way QuickSort, an array arr[l..r] is divided in 3 parts:

- a) arr[i..i] elements less than pivot.
- b) arr[i+1..j-1] elements equal to pivot.
- c) arr[j..r] elements greater than pivot.

See [this](#) for implementation.

How to implement QuickSort for Linked Lists?

[QuickSort on Singly Linked List](#)

[QuickSort on Doubly Linked List](#)

Can we implement QuickSort Iteratively?

Yes, please refer [Iterative Quick Sort](#).

Why Quick Sort is preferred over MergeSort for sorting Arrays

Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires $O(N)$ extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have $O(N\log N)$ average complexity but the constants differ. For arrays, merge sort loses due to the use of extra $O(N)$ storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of $O(n\log n)$. The worst case is possible in randomized version also, but worst case doesn't occur for a

particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.

Quick Sort is also tail recursive, therefore tail call optimizations is done.

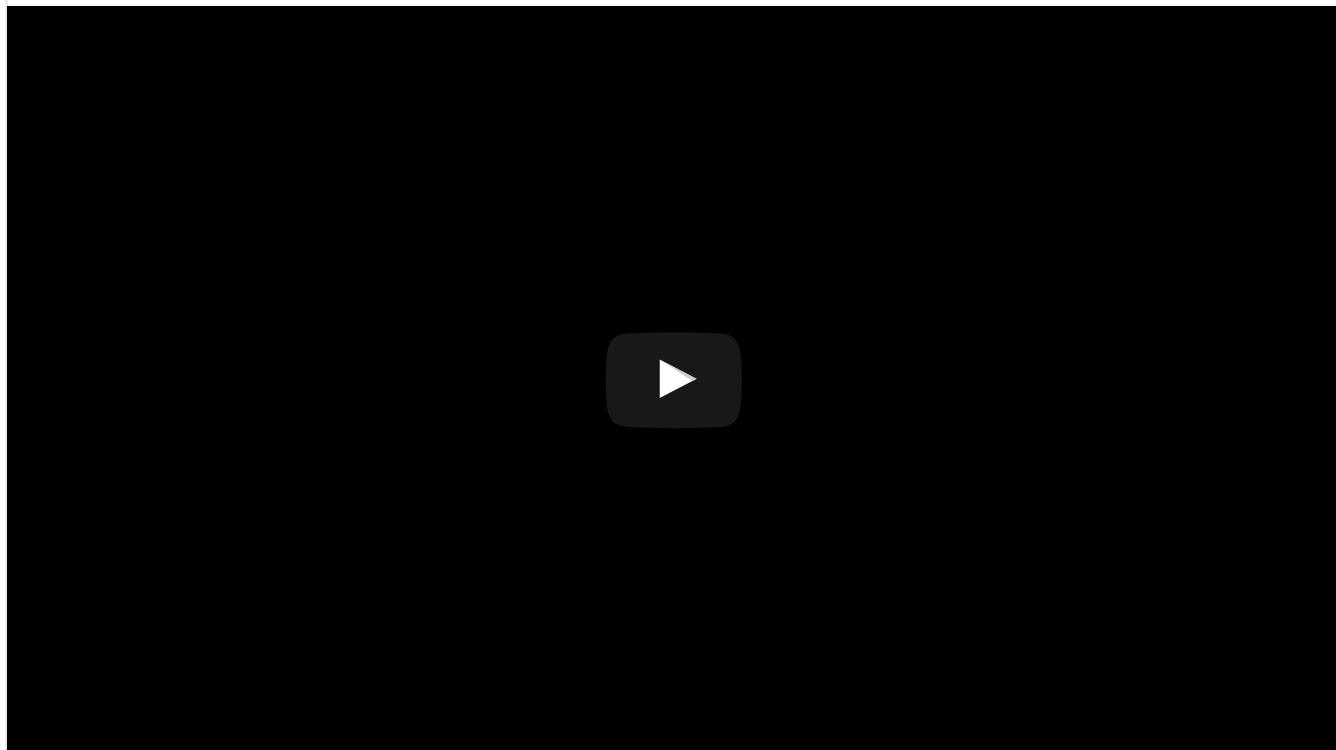
Why MergeSort is preferred over QuickSort for Linked Lists?

In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of $A[0]$ be x then to access $A[i]$, we can directly access the memory at $(x + i \cdot 4)$. Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i 'th index, we have to travel each and every node from the head to i 'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

How to optimize QuickSort so that it takes $O(\log n)$ extra space in worst case?

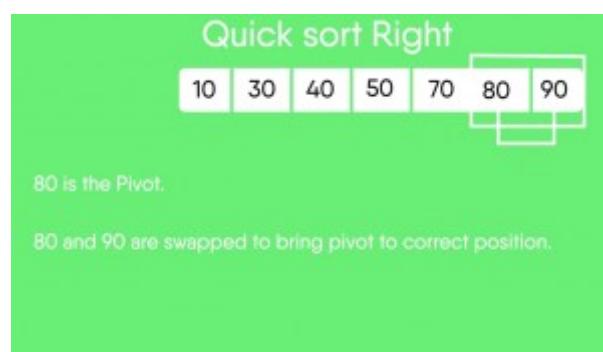
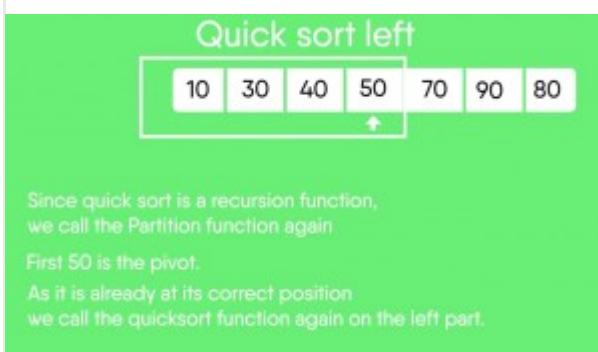
Please see [QuickSort Tail Call Optimization \(Reducing worst case space to \$\log n\$ \)](#)



Snapshots:

Partition						
10	80	30	90	40	50	70
Counter variables						Pivot
I: Index of smaller element						We start the loop with initial values.
J: Loop variable						
Test condition		Actions		Value of variables		
$arr[J] \leq pivot$				$I = 1$ $J = 0$		

Partition						
10	80	30	90	40	50	70
Counter variables						Pivot
I: Index of smaller element						
J: Loop variable						
Test condition		Actions		Value of variables		
$arr[J] \leq pivot$				$I = 0$ $J = 1$		
80 < 70 False		No action				
Pass 2						



Quiz on QuickSort

References:

<http://en.wikipedia.org/wiki/Quicksort>

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort, Pigeonhole Sort

Coding practice for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner



Download

Zip, Unzip or Open Any File
With Unzipper.

Unzipper



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Searching and Sorting

([Login](#) to Rate and Mark)

Average Rating : **4.8/5.0**

Based on **5** vote(s)



Average Difficulty : **3.2/5.0**

3.2

Based on **46** vote(s)



Add to TODO List

Mark as DONE

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Radix Sort

The **lower bound for Comparison based sorting algorithm** (Merge Sort, Heap Sort, Quick-Sort .. etc) is $\Omega(n \log n)$, i.e., they cannot do better than $n \log n$.

Counting sort is a linear time sorting algorithm that sorts in $O(n+k)$ time when elements are in range from 1 to k .

What if the elements are in range from 1 to n^2 ?

We can't use counting sort because counting sort will take $O(n^2)$ which is worse than comparison based sorting algorithms. Can we sort such an array in linear time?

Radix Sort is the answer. The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

The Radix Sort Algorithm

1) Do following for each digit i where i varies from least significant digit to the most significant digit.

.....a) Sort input array using counting sort (or any stable sort) according to the i 'th digit.

Example:

Original, unsorted list:

170, 45, 75, 90, 802, 24, 2, 66

Sorting by least significant digit (1s place) gives: [*Notice that we keep 802 before 2, because 802 occurred before 2 in the original list, and similarly for pairs 170 & 90 and 45 & 75.]

170, 90, 802, 2, 24, 45, 75, 66

Sorting by next digit (10s place) gives: [*Notice that 802 again comes before 2 as 802 comes before 2 in the previous list.]

802, 2, 24, 45, 66, 170, 75, 90

Sorting by most significant digit (100s place) gives:

2, 24, 45, 66, 75, 90, 170, 802

What is the running time of Radix Sort?

Let there be d digits in input integers. Radix Sort takes $O(d*(n+b))$ time where b is the base for representing numbers, for example, for decimal system, b is 10. What is the value of d ? If k is the maximum possible value, then d would be $O(\log_b(k))$. So overall time complexity is $O((n+b) * \log_b(k))$. Which looks more than the time complexity of comparison based sorting algorithms for a large k . Let us first limit k . Let $k \leq n^c$ where c is a

constant. In that case, the complexity becomes $O(n\log_b(n))$. But it still doesn't beat comparison based sorting algorithms.

What if we make value of b larger?. What should be the value of b to make the time complexity linear? If we set b as n, we get the time complexity as $O(n)$. In other words, we can sort an array of integers with range from 1 to n^c if the numbers are represented in base n (or every digit takes $\log_2(n)$ bits).

Is Radix Sort preferable to Comparison based sorting algorithms like Quick-Sort?

If we have $\log_2 n$ bits for every digit, the running time of Radix appears to be better than Quick Sort for a wide range of input numbers. The constant factors hidden in asymptotic notation are higher for Radix Sort and Quick-Sort uses hardware caches more effectively. Also, Radix sort uses counting sort as a subroutine and counting sort takes extra space to sort numbers.

Implementation of Radix Sort

Following is a simple C++ implementation of Radix Sort. For simplicity, the value of d is assumed to be 10. We recommend you to see [Counting Sort](#) for details of countSort() function in below code.

```
// C++ implementation of Radix Sort
#include<iostream>
using namespace std;

// A utility function to get maximum value in arr[]
int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp)
{
    int output[n]; // output array
    int i, count[10] = {0};

    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
        count[(arr[i]/exp)%10]++;
    
    // Change count[i] so that count[i] now contains actual
    // position of this digit in output[]
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i]/exp)%10] - 1] = arr[i];
        count[(arr[i]/exp)%10]--;
    }

    // Copy the output array to arr[], so that arr[] now
    // contains sorted numbers according to current digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n)
{
```

```

// Find the maximum number to know number of digits
int m = getMax(arr, n);

// Do counting sort for every digit. Note that instead
// of passing digit number, exp is passed. exp is 10^i
// where i is current digit number
for (int exp = 1; m/exp > 0; exp *= 10)
    countSort(arr, n, exp);
}

// A utility function to print an array
void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above functions
int main()
{
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr)/sizeof(arr[0]);
    radixsort(arr, n);
    print(arr, n);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Radix sort Java implementation
import java.io.*;
import java.util.*;

class Radix {

    // A utility function to get maximum value in arr[]
    static int getMax(int arr[], int n)
    {
        int mx = arr[0];
        for (int i = 1; i < n; i++)
            if (arr[i] > mx)
                mx = arr[i];
        return mx;
    }

    // A function to do counting sort of arr[] according to
    // the digit represented by exp.
    static void countSort(int arr[], int n, int exp)
    {
        int output[] = new int[n]; // output array
        int i;
        int count[] = new int[10];
        Arrays.fill(count, 0);

        // Store count of occurrences in count[]
        for (i = 0; i < n; i++)
            count[ (arr[i]/exp)%10 ]++;

        // Change count[i] so that count[i] now contains
        // actual position of this digit in output[]
        for (i = 1; i < 10; i++)
            count[i] += count[i - 1];

        // Build the output array
        for (i = n - 1; i >= 0; i--)
        {
            output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];
            count[ (arr[i]/exp)%10 ]--;
        }
    }
}

```

```

// Copy the output array to arr[], so that arr[] now
// contains sorted numbers according to current digit
for (i = 0; i < n; i++)
    arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
static void radixsort(int arr[], int n)
{
    // Find the maximum number to know number of digits
    int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (int exp = 1; m/exp > 0; exp *= 10)
        countSort(arr, n, exp);
}

// A utility function to print an array
static void print(int arr[], int n)
{
    for (int i=0; i<n; i++)
        System.out.print(arr[i]+ " ");
}

/*Driver function to check for above function*/
public static void main (String[] args)
{
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = arr.length;
    radixsort(arr, n);
    print(arr, n);
}
/* This code is contributed by Devesh Agrawal */

```

[Run on IDE](#)

Python

```

# Python program for implementation of Radix Sort

# A function to do counting sort of arr[] according to
# the digit represented by exp.
def countingSort(arr, exp1):

    n = len(arr)

    # The output array elements that will have sorted arr
    output = [0] * (n)

    # Initialize count array as 0
    count = [0] * (10)

    # Store count of occurrences in count[]
    for i in range(0, n):
        index = (arr[i]/exp1)
        count[ (index)%10 ] += 1

    # Change count[i] so that count[i] now contains actual
    # position of this digit in output array
    for i in range(1,10):
        count[i] += count[i-1]

    # Build the output array
    i = n-1
    while i>=0:

```

```
index = (arr[i]/exp1)
output[ count[ (index)%10 ] - 1] = arr[i]
count[ (index)%10 ] -= 1
i -= 1

# Copying the output array to arr[],
# so that arr now contains sorted numbers
i = 0
for i in range(0,len(arr)):
    arr[i] = output[i]

# Method to do Radix Sort
def radixSort(arr):

    # Find the maximum number to know number of digits
    max1 = max(arr)

    # Do counting sort for every digit. Note that instead
    # of passing digit number, exp is passed. exp is 10^i
    # where i is current digit number
    exp = 1
    while max1/exp > 0:
        countingSort(arr,exp)
        exp *= 10

# Driver code to test above
arr = [ 170, 45, 75, 90, 802, 24, 2, 66]
radixSort(arr)

for i in range(len(arr)):
    print(arr[i]),

# This code is contributed by Mohit Kumra
```

[Run on IDE](#)

Output:

```
2 24 45 66 75 90 170 802
```



Snapshots:

Consider this input array

170	45	75	90	802	24	2	66
-----	----	----	----	-----	----	---	----

First consider the one's place

Consider this input array

170	45	75	90	802	24	2	66
-----	----	----	----	-----	----	---	----

170	90	802	2	24	45	75	66
-----	----	-----	---	----	----	----	----

Consider this input array

170	45	75	90	802	24	2	66
170	90	802	2	24	45	75	66

Observe that 170 has come before 90 this is because it appeared before in the original list.

Consider this input array

170	45	75	90	802	24	2	66
170	90	802	2	24	45	75	66
802	2	24	45	66	170	75	90

Now consider the 100's place.

Array is Now sorted

2	24	45	66	75	90	170	802
---	----	----	----	----	----	-----	-----

Quiz on Radix Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Counting Sort
- Bucket Sort
- ShellSort

References:

http://en.wikipedia.org/wiki/Radix_sort

<http://alg12.wikischolars.columbia.edu/file/view/RADIX.pdf>

MIT Video Lecture 

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Sorting

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 36 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

Practice

GATE CS

Placements

GeeksQuiz

Google™ Custom Search



Login/Register

Comb Sort

Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all inversions are removed one by one. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one inversion counts with one swap and performs better than Bubble Sort.

The shrink factor has been empirically found to be 1.3 (by testing Combsort on over 200,000 random lists) [Source: [Wiki](#)]

Although, it works better than Bubble Sort on average, worst case remains $O(n^2)$.

Below is C++ implementation.

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

// To find gap between elements
int getNextGap(int gap)
{
    // Shrink gap by Shrink factor
    gap = (gap*10)/13;

    if (gap < 1)
        return 1;
    return gap;
}

// Function to sort a[0..n-1] using Comb Sort
void combSort(int a[], int n)
{
    // Initialize gap
    int gap = n;

    // Initialize swapped as true to make sure that
    // loop runs
    bool swapped = true;

    // Keep running while gap is more than 1 and last
    // iteration caused a swap
    while (gap != 1 || swapped == true)
    {
        // Find next gap
        gap = getNextGap(gap);

        // Initialize swapped as false so that we can
        // check if swap happened or not
        swapped = false;

        for (int i = 0; i + gap < n; i += gap)
        {
            if (a[i] > a[i + gap])
            {
                swap(a[i], a[i + gap]);
                swapped = true;
            }
        }
    }
}
```

```

// Compare all elements with current gap
for (int i=0; i<n-gap; i++)
{
    if (a[i] > a[i+gap])
    {
        swap(a[i], a[i+gap]);
        swapped = true;
    }
}
}

// Driver program
int main()
{
    int a[] = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
    int n = sizeof(a)/sizeof(a[0]);

    combSort(a, n);

    printf("Sorted array: \n");
    for (int i=0; i<n; i++)
        printf("%d ", a[i]);

    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program for implementation of Comb Sort
class CombSort
{
    // To find gap between elements
    int getNextGap(int gap)
    {
        // Shrink gap by Shrink factor
        gap = (gap*10)/13;
        if (gap < 1)
            return 1;
        return gap;
    }

    // Function to sort arr[] using Comb Sort
    void sort(int arr[])
    {
        int n = arr.length;

        // initialize gap
        int gap = n;

        // Initialize swapped as true to make sure that
        // loop runs
        boolean swapped = true;

        // Keep running while gap is more than 1 and last
        // iteration caused a swap
        while (gap != 1 || swapped == true)
        {
            // Find next gap
            gap = getNextGap(gap);

            // Initialize swapped as false so that we can
            // check if swap happened or not
            swapped = false;

            // Compare all elements with current gap
            for (int i=0; i<n-gap; i++)
            {
                if (arr[i] > arr[i+gap])

```

```

        {
            // Swap arr[i] and arr[i+gap]
            int temp = arr[i];
            arr[i] = arr[i+gap];
            arr[i+gap] = temp;

            // Set swapped
            swapped = true;
        }
    }
}

// Driver method
public static void main(String args[])
{
    CombSort ob = new CombSort();
    int arr[] = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
    ob.sort(arr);

    System.out.println("sorted array");
    for (int i=0; i<arr.length; ++i)
        System.out.print(arr[i] + " ");
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# Python program for implementation of CombSort

# To find next gap from current
def getNextGap(gap):

    # Shrink gap by Shrink factor
    gap = (gap * 10)/13
    if gap < 1:
        return 1
    return gap

# Function to sort arr[] using Comb Sort
def combSort(arr):
    n = len(arr)

    # Initialize gap
    gap = n

    # Initialize swapped as true to make sure that
    # loop runs
    swapped = True

    # Keep running while gap is more than 1 and last
    # iteration caused a swap
    while gap !=1 or swapped == 1:

        # Find next gap
        gap = getNextGap(gap)

        # Initialize swapped as false so that we can
        # check if swap happened or not
        swapped = False

        # Compare all elements with current gap
        for i in range(0, n-gap):
            if arr[i] > arr[i + gap]:
                arr[i], arr[i + gap]=arr[i + gap], arr[i]
                swapped = True

```

```
# Driver code to test above
arr = [ 8, 4, 1, 3, -44, 23, -6, 28, 0]
combSort(arr)

print ("Sorted array:")
for i in range(len(arr)):
    print (arr[i]),

# This code is contributed by Mohit Kumra
```

[Run on IDE](#)

Output :

```
Sorted array:
-44 -6 0 1 3 4 8 23 28 56
```

Illustration:

Let the array elements be

```
8, 4, 1, 56, 3, -44, 23, -6, 28, 0
```

Initially gap value = 10

After shrinking gap value => $10/1.3 = 7$;

```
8 4 1 56 3 -44 23 -6 28 0
-6 4 1 56 3 -44 23 8 28 0
-6 4 0 56 3 -44 23 8 28 1
```

New gap value => $7/1.3 = 5$;

```
-44 4 0 56 3 -6 23 8 28 1
-44 4 0 28 3 -6 23 8 56 1
-44 4 0 28 1 -6 23 8 56 3
```

New gap value => $5/1.3 = 3$;

```
-44 1 0 28 4 -6 23 8 56 3
-44 1 -6 28 4 0 23 8 56 3
-44 1 -6 23 4 0 28 8 56 3
-44 1 -6 23 4 0 3 8 56 28
```

New gap value => $3/1.3 = 2$;

```
-44 1 -6 0 4 23 3 8 56 28
-44 1 -6 0 3 23 4 8 56 28
-44 1 -6 0 3 8 4 23 56 28
```

New gap value => $2/1.3 = 1$;

```
-44 -6 1 0 3 8 4 23 56 28
-44 -6 0 1 3 8 4 23 56 28
-44 -6 0 1 3 4 8 23 56 28
-44 -6 0 1 3 4 8 23 28 56
```

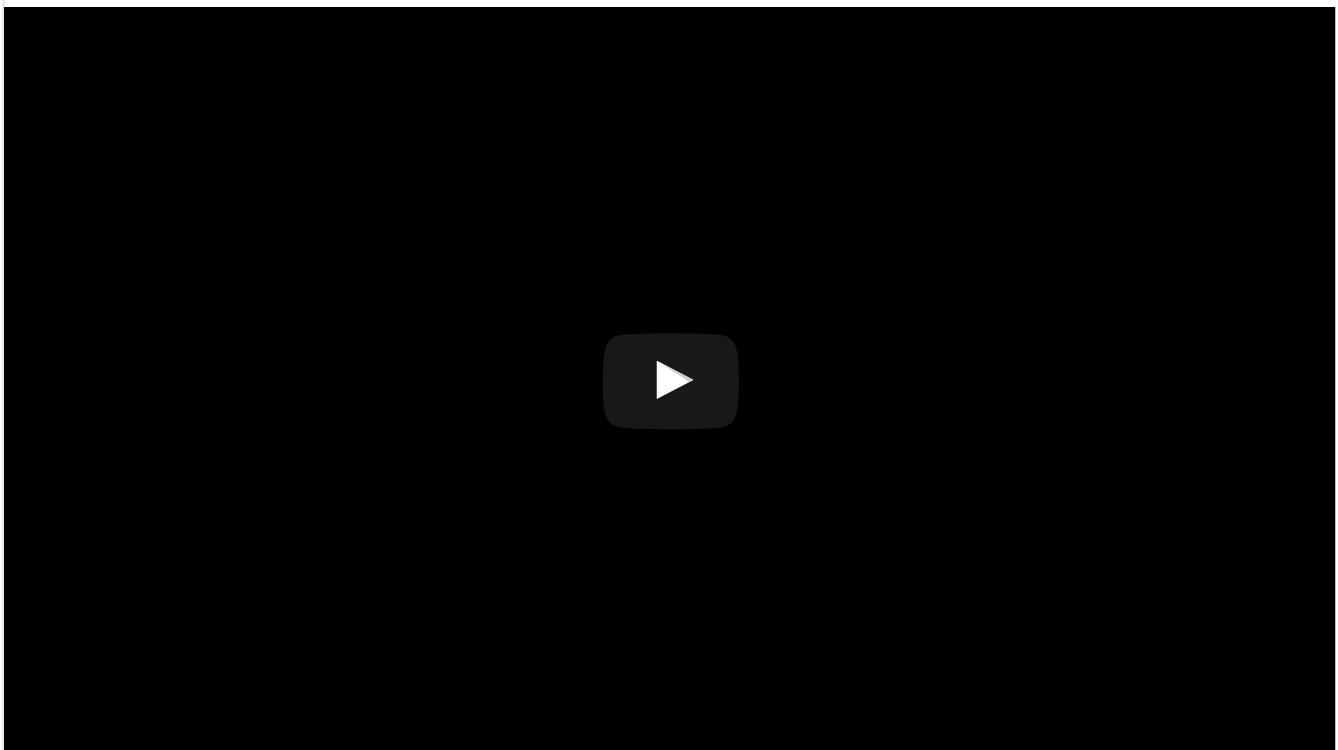
no more swaps required (Array sorted)

Time Complexity : Worst case complexity of this algorithm is $O(n^2)$ and the Best Case complexity is $O(n)$.

Auxiliary Space : $O(1)$.

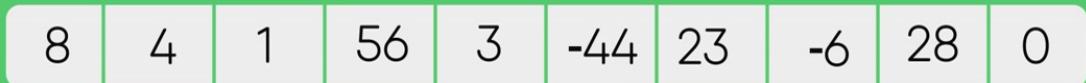
Quiz on Comb Sort

This article is contributed by **Rahul Agrawal**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.



Snapshots:

Let the array elements be



Gap value

10

Run No

1

Comments

No change

Let the array elements be



Gap value

$10/1.3 = 7$

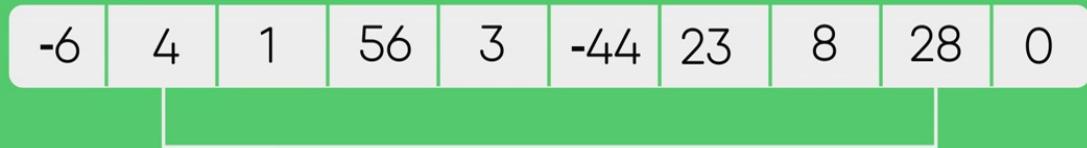
Run No

2

Comments

Compare and swap values

Let the array elements be



Gap value

$$10/1.3 = 7$$

Run No

2

Comments

Compare and swap values

Let the array elements be



Gap value

$$2/1.3 = 1$$

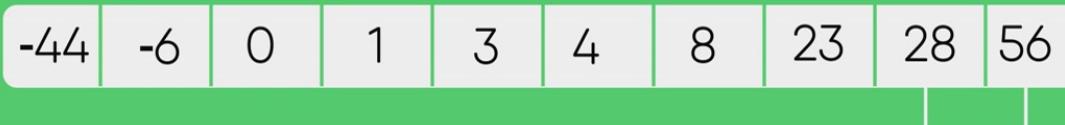
Run No

6

Comments

Compare and swap values

Let the array elements be



Gap value

$$2/1.3 = 1$$

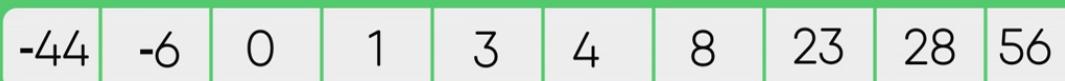
Run No

6

Comments

Compare and swap values

Let the array elements be



Array is now sorted

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz

[Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#), [Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Pigeonhole Sort](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

Sorting

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 12 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Pigeonhole Sort

Pigeonhole sorting is a sorting algorithm that is suitable for sorting lists of elements where the number of elements and the number of possible key values are approximately the same.

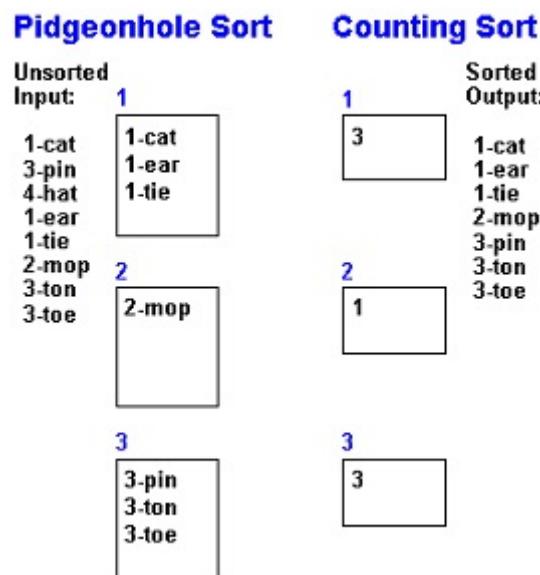
It requires $O(n + \text{Range})$ time where n is number of elements in input array and ‘Range’ is number of possible values in array.

Working of Algorithm :

- Find minimum and maximum values in array. Let the minimum and maximum values be ‘min’ and ‘max’ respectively. Also find range as ‘max-min-1’.
- Set up an array of initially empty “pigeonholes” the same size as of the range.
- Visit each element of the array and then put each element in its pigeonhole. An element arr[i] is put in hole at index arr[i] – min.
- Start the loop all over the pigeonhole array in order and put the elements from non- empty holes back into the original array.

Comparison with Counting Sort :

It is similar to [counting sort](#), but differs in that it “moves items twice: once to the bucket array and again to the final destination” .



Below is C++ implementation of Pigeonhole Sort.

```
/* C program to implement Pigeonhole Sort */
#include <bits/stdc++.h>
```

```

using namespace std;

/* Sorts the array using pigeonhole algorithm */
void pigeonholeSort(int arr[], int n)
{
    // Find minimum and maximum values in arr[]
    int min = arr[0], max = arr[0];
    for (int i = 1; i < n; i++)
    {
        if (arr[i] < min)
            min = arr[i];
        if (arr[i] > max)
            max = arr[i];
    }
    int range = max - min + 1; // Find range

    // Create an array of vectors. Size of array
    // range. Each vector represents a hole that
    // is going to contain matching elements.
    vector<int> holes[range];

    // Traverse through input array and put every
    // element in its respective hole
    for (int i = 0; i < n; i++)
        holes[arr[i]-min].push_back(arr[i]);

    // Traverse through all holes one by one. For
    // every hole, take its elements and put in
    // array.
    int index = 0; // index in sorted array
    for (int i = 0; i < range; i++)
    {
        vector<int>::iterator it;
        for (it = holes[i].begin(); it != holes[i].end(); ++it)
            arr[index++] = *it;
    }
}

// Driver program to test the above function
int main()
{
    int arr[] = {8, 3, 2, 7, 4, 6, 8};
    int n = sizeof(arr)/sizeof(arr[0]);

    pigeonholeSort(arr, n);

    printf("Sorted order is : ");
    for (int i = 0; i < n; i++)
        printf(" %d ", arr[i]);

    return 0;
}

```

Run on IDE

Output:

```
Sorted order is : 2 3 4 6 7 8 8
```

Pigeonhole sort has limited use as requirements are rarely met. For arrays where range is much larger than n , bucket sort is a generalization that is more efficient in space and time.

References:

https://en.wikipedia.org/wiki/Pigeonhole_sort

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

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz

[Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#), [Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Comb Sort](#),

The advertisement features a dark brown header with white text. At the top right is a small blue info icon and a close button. Below the header, the title "Introduction to Computer Science" is displayed in large white font. Underneath the title, the text "ENROLLMENT IS OPEN TO EVERYONE" is shown in smaller white font. In the center is a white rectangular button with a black border and the text "Start Now" in bold black font. Below the button, there is a horizontal line separator. On the left side of the line is the edX logo (a stylized "e" and "X" in red and blue) and the website address "www.edx.org". On the right side is the Harvard University logo (a crest with three stars and the word "HARVARD") and the text "HARVARD UNIVERSITY".

GATE CS Corner Company Wise Coding Practice

Sorting

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2

Average Difficulty : 2/5.0
Based on 12 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Interpolation search vs Binary search

[Interpolation search](#) works better than Binary Search for a sorted and uniformly distributed array.

On average the interpolation search makes about $\log(\log(n))$ comparisons (if the elements are uniformly distributed), where n is the number of elements to be searched. In the worst case (for instance where the numerical values of the keys increase exponentially) it can make up to O(n) comparisons.

[Interpolation Search Article](#)

Sources:

http://en.wikipedia.org/wiki/Interpolation_search



GATE CS Corner Company Wise Coding Practice

[Searching](#) [Binary-Search](#)

Related Posts:

- Second minimum element using minimum comparisons
- [Interpolation Search](#)
- [Jump Search](#)
- Third largest element in an array of distinct elements

- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

(Login to Rate and Mark)

2.2

Average Difficulty : **2.2/5.0**
Based on **26** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Stability in sorting algorithms

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array. Some sorting algorithms are stable by nature like Insertion sort, Merge Sort, Bubble Sort, etc. And some sorting algorithms are not, like Heap Sort, Quick Sort, etc.

However, any given sorting algo which is not stable can be modified to be stable. There can be sorting algo specific ways to make it stable, but in general, any comparison based sorting algorithm which is not stable by nature can be modified to be stable by changing the key comparison operation so that the comparison of two keys considers position as a factor for objects with equal keys.

References:

<http://www.math.uic.edu/~leon/cs-mcs401-s08/handouts/stability.pdf>

http://en.wikipedia.org/wiki/Sorting_algorithm#Stability

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

✖
Introduction to
Computer Science

ENROLLMENT IS OPEN TO EVERYONE

[Start Now](#)

 www.edx.org
 HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Sorting](#) [Merge Sort](#) [Quick Sort](#)

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

1.4

Average Difficulty : **1.4/5.0**
Based on **34** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

When does the worst case of Quicksort occur?

The answer depends on strategy for choosing pivot. In early versions of Quick Sort where leftmost (or rightmost) element is chosen as pivot, the worst occurs in following cases.

- 1) Array is already sorted in same order.
- 2) Array is already sorted in reverse order.
- 3) All elements are same (special case of case 1 and 2)

Since these cases are very common use cases, the problem was easily solved by choosing either a random index for the pivot, choosing the middle index of the partition or (especially for longer partitions) choosing the median of the first, middle and last element of the partition for the pivot. With these modifications, the worst case of Quick sort has less chances to occur, but worst case can still occur if the input array is such that the maximum (or minimum) element is always chosen as pivot.

References:

<http://en.wikipedia.org/wiki/Quicksort>

(i) X

Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now

edX www.edx.org

HARVARD UNIVERSITY

GATE CS Corner Company Wise Coding Practice

Sorting Quick Sort

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

1.4

Average Difficulty : **1.4/5.0**
Based on **24** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Lower bound for comparison based sorting algorithms

The problem of sorting can be viewed as following.

Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \dots \leq a'_n$.

A sorting algorithm is comparison based if it uses comparison operators to find the order between two numbers. Comparison sorts can be viewed abstractly in terms of decision trees. A decision tree is a **full binary tree** that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size. The execution of the sorting algorithm corresponds to tracing a path from the root of the decision tree to a leaf. At each internal node, a comparison $a_i \leq a_j$ is made. The left subtree then dictates subsequent comparisons for $a_i \leq a_j$, and the right subtree dictates subsequent comparisons for $a_i > a_j$. When we come to a leaf, the sorting algorithm has established the ordering. So we can say following about the decision tree.

1) Each of the $n!$ permutations on n elements must appear as one of the leaves of the decision tree for the sorting algorithm to sort properly.

2) Let x be the maximum number of comparisons in a sorting algorithm. The maximum height of the decision tree would be x . A tree with maximum height x has at most 2^x leaves.

After combining the above two facts, we get following relation.

$$n! \leq 2^x$$

Taking Log on both sides.

$$\log_2(n!) \leq x$$

Since $\log_2(n!) = \Theta(n \log n)$, we can say

$$x = \Omega(n \log n)$$

Therefore, any comparison based sorting algorithm must make at least $n \log n$ comparisons to sort the input array, and Heapsort and merge sort are asymptotically optimal comparison sorts.

References:

Introduction to Algorithms, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Sorting](#) [Merge Sort](#)

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2.3

Average Difficulty : 2.3/5.0
Based on 12 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Which sorting algorithm makes minimum number of memory writes?

Minimizing the number of writes is useful when making writes to some huge data set is very expensive, such as with [EEPROMs](#) or [Flash memory](#), where each write reduces the lifespan of the memory.

Among the sorting algorithms that we generally study in our data structure and algorithm courses, [Selection Sort](#) makes least number of writes (it makes $O(n)$ swaps). But, [Cycle Sort](#) almost always makes less number of writes compared to Selection Sort. In Cycle Sort, each value is either written zero times, if it's already in its correct position, or written one time to its correct position. This matches the minimal number of overwrites required for a completed in-place sort.

Sources:

http://en.wikipedia.org/wiki/Cycle_sort

http://en.wikipedia.org/wiki/Selection_sort

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

ⓘ ✕

Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



www.edx.org


HARVARD
UNIVERSITY

[GATE CS Corner](#) [Company Wise Coding Practice](#)

Sorting

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2.4

Average Difficulty : **2.4/5.0**
Based on 31 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

Practice

GATE CS

Placements

GeeksQuiz

Google™ Custom Search



Login/Register

Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted

Given an unsorted array $\text{arr}[0..n-1]$ of size n , find the minimum length subarray $\text{arr}[s..e]$ such that sorting this subarray makes the whole array sorted.

Examples:

- 1) If the input array is [10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60], your program should be able to find that the subarray lies between the indexes 3 and 8.
- 2) If the input array is [0, 1, 15, 25, 6, 7, 30, 40, 50], your program should be able to find that the subarray lies between the indexes 2 and 5.

We strongly recommend that you click here and practice it, before moving on to the solution.

Solution:

1) Find the candidate unsorted subarray

- a) Scan from left to right and find the first element which is greater than the next element. Let s be the index of such an element. In the above example 1, s is 3 (index of 30).
- b) Scan from right to left and find the first element (first in right to left order) which is smaller than the next element (next in right to left order). Let e be the index of such an element. In the above example 1, e is 7 (index of 31).

2) Check whether sorting the candidate unsorted subarray makes the complete array sorted or not. If not, then include more elements in the subarray.

- a) Find the minimum and maximum values in $\text{arr}[s..e]$. Let minimum and maximum values be min and max . min and max for [30, 25, 40, 32, 31] are 25 and 40 respectively.
- b) Find the first element (if there is any) in $\text{arr}[0..s-1]$ which is greater than min , change s to index of this element. There is no such element in above example 1.
- c) Find the last element (if there is any) in $\text{arr}[e+1..n-1]$ which is smaller than max , change e to index of this element. In the above example 1, e is changed to 8 (index of 35)

3) Print s and e .

Implementation:

```
#include<stdio.h>

void printUnsorted(int arr[], int n)
{
    int s = 0, e = n-1, i, max, min;

    // step 1(a) of above algo
    for (s = 0; s < n-1; s++)
    {
        if (arr[s] > arr[s+1])
            break;
    }
    if (s == n-1)
    {
        printf("The complete array is sorted");
        return;
    }

    // step 1(b) of above algo
    for(e = n - 1; e > 0; e--)
    {
        if(arr[e] < arr[e-1])
            break;
    }

    // step 2(a) of above algo
    max = arr[s]; min = arr[s];
    for(i = s + 1; i <= e; i++)
    {
        if(arr[i] > max)
            max = arr[i];
        if(arr[i] < min)
            min = arr[i];
    }

    // step 2(b) of above algo
    for( i = 0; i < s; i++)
    {
        if(arr[i] > min)
        {
            s = i;
            break;
        }
    }

    // step 2(c) of above algo
    for( i = n - 1; i >= e+1; i--)
    {
        if(arr[i] < max)
        {
            e = i;
            break;
        }
    }

    // step 3 of above algo
    printf(" The unsorted subarray which makes the given array "
           " sorted lies between the indees %d and %d", s, e);
    return;
}

int main()
{
    int arr[] = {10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
    printUnsorted(arr, arr_size);
    getchar();
    return 0;
}
```

Run on IDE

Time Complexity: O(n)

Please write comments if you find the above code/algorithm incorrect, or find better ways to solve the same problem.



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Sorting

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 79 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Merge Sort for Linked Lists

Merge sort is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

Let head be the first node of the linked list to be sorted and headRef be the pointer to head. Note that we need a reference to head in MergeSort() as the below implementation changes next links to sort the linked lists (not data at the nodes), so head node has to be changed if the data at original head is not the smallest value in linked list.

```

MergeSort(headRef)
1) If head is NULL or there is only one element in the Linked List
   then return.
2) Else divide the linked list into two halves.
   FrontBackSplit(head, &a, &b); /* a and b are two halves */
3) Sort the two halves a and b.
   MergeSort(a);
   MergeSort(b);
4) Merge the sorted a and b (using SortedMerge() discussed here)
   and update the head pointer using headRef.
   *headRef = SortedMerge(a, b);

```

```

#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* function prototypes */
struct node* SortedMerge(struct node* a, struct node* b);
void FrontBackSplit(struct node* source,
                    struct node** frontRef, struct node** backRef);

/* sorts the linked list by changing next pointers (not data) */
void MergeSort(struct node** headRef)
{
    struct node* head = *headRef;
    struct node* a;
    struct node* 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);
}

/* See http://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);
}

/* 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;
    }
}

```

```

}

/* Function to print nodes in a given linked list */
void printList(struct node *node)
{
    while(node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Function to insert a node at the begining 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 off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct node* res = NULL;
    struct node* a = NULL;

    /* Let us create a unsorted linked lists to test the functions
     * Created lists shall be a: 2->3->20->5->10->15 */
    push(&a, 15);
    push(&a, 10);
    push(&a, 5);
    push(&a, 20);
    push(&a, 3);
    push(&a, 2);

    /* Sort the above created Linked List */
    MergeSort(&a);

    printf("\n Sorted Linked List is: \n");
    printList(a);

    getchar();
    return 0;
}

```

Run on IDE

Time Complexity: O(nLogn)

Sources:

http://en.wikipedia.org/wiki/Merge_sort

<http://cslibrary.stanford.edu/105/LinkedListProblems.pdf>

Please write comments if you find the above code/algorithm incorrect, or find better ways to solve the same problem.



Download

Zip, Unzip or Open Any File
With Unzipper.

Unzipper



GATE CS Corner Company Wise Coding Practice

[Linked Lists](#) [Sorting](#) [Merge Sort](#)

Related Posts:

- Iteratively Reverse a linked list using only 2 pointers (An Interesting Method)
- Find pair for given sum in a sorted singly linked without extra space
- Find pairs with given sum in doubly linked list
- Unrolled Linked List | Set 1 (Introduction)
- Convert a Binary Tree to a Circular Doubly Link List
- Subtract Two Numbers represented as Linked Lists
- Rearrange a given list such that it consists of alternating minimum maximum elements
- Flatten a multi-level linked list | Set 2 (Depth wise)

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 149 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Sort a nearly sorted (or K sorted) array

Given an array of n elements, where each element is at most k away from its target position, devise an algorithm that sorts in $O(n \log k)$ time.

For example, let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.

Source: [Nearly sorted algorithm](#)

We can use **Insertion Sort** to sort the elements efficiently. Following is the C code for standard Insertion Sort.

```
/* Function to sort an array using insertion sort*/
void insertionSort(int A[], int size)
{
    int i, key, j;
    for (i = 1; i < size; i++)
    {
        key = A[i];
        j = i-1;

        /* Move elements of A[0..i-1], that are greater than key, to one
           position ahead of their current position.
           This loop will run at most k times */
        while (j >= 0 && A[j] > key)
        {
            A[j+1] = A[j];
            j = j-1;
        }
        A[j+1] = key;
    }
}
```

[Run on IDE](#)

The inner loop will run at most k times. To move every element to its correct place, at most k elements need to be moved. So overall complexity will be $O(nk)$

We can sort such arrays **more efficiently with the help of Heap data structure**. Following is the detailed process that uses Heap.

- 1) Create a Min Heap of size $k+1$ with first $k+1$ elements. This will take $O(k)$ time (See [this GFact](#))
- 2) One by one remove min element from heap, put it in result array, and add a new element to heap from remaining elements.

Removing an element and adding a new element to min heap will take LogK time. So overall complexity will be $O(k) + O((n-k)*\log K)$

```
#include<iostream>
using namespace std;
```

```

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor
    MinHeap(int 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 remove min (or root), add a new value x, and return old root
    int replaceMin(int x);

    // to extract the root which is the minimum element
    int extractMin();
};

// Given an array of size n, where every element is k away from its target
// position, sorts the array in O(nLogk) time.
int sortK(int arr[], int n, int k)
{
    // Create a Min Heap of first (k+1) elements from
    // input array
    int *harr = new int[k+1];
    for (int i = 0; i<=k && i<n; i++) // i < n condition is needed when k > n
        harr[i] = arr[i];
    MinHeap hp(harr, k+1);

    // i is index for remaining elements in arr[] and ti
    // is target index of for cuurent minimum element in
    // Min Heapm 'hp'.
    for(int i = k+1, ti = 0; ti < n; i++, ti++)
    {
        // If there are remaining elements, then place
        // root of heap at target index and add arr[i]
        // to Min Heap
        if (i < n)
            arr[ti] = hp.replaceMin(arr[i]);

        // Otherwise place root at its target index and
        // reduce heap size
        else
            arr[ti] = hp.extractMin();
    }
}

// FOLLOWING ARE IMPLEMENTATIONS OF STANDARD MIN HEAP METHODS FROM CORMEN BOOK
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{

```

```

int root = harr[0];
if (heap_size > 1)
{
    harr[0] = harr[heap_size-1];
    heap_size--;
    MinHeapify(0);
}
return root;
}

// Method to change root with given value x, and return the old root
int MinHeap::replaceMin(int x)
{
    int root = harr[0];
    harr[0] = x;
    if (root < x)
        MinHeapify(0);
    return root;
}

// 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] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// A utility function to print array elements
void printArray(int arr[], int size)
{
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    int k = 3;
    int arr[] = {2, 6, 3, 12, 56, 8};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortK(arr, n, k);

    cout << "Following is sorted array\n";
    printArray (arr, n);

    return 0;
}

```

Run on IDE

Output:

Following is sorted array

2 3 6 8 12 56

The Min Heap based method takes $O(n\log k)$ time and uses $O(k)$ auxiliary space.

We can also **use a Balanced Binary Search Tree** instead of Heap to store $K+1$ elements. The **insert** and **delete** operations on Balanced BST also take $O(\log k)$ time. So Balanced BST based method will also take $O(n\log k)$ time, but the Heap based method seems to be more efficient as the minimum element will always be at root. Also, Heap doesn't need extra space for left and right pointers.

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.



GATE CS Corner Company Wise Coding Practice

[Heap](#) [Sorting](#) [Insertion Sort](#)

Related Posts:

- Sum of all elements between $k1^{\text{th}}$ and $k2^{\text{th}}$ smallest elements
- Minimum sum of two numbers formed from digits of an array
- Heap using STL C++
- Convert min Heap to max Heap
- K-ary Heap
- Check if a given Binary Tree is Heap
- Fibonacci Heap | Set 1 (Introduction)
- How to check if a given array represents a Binary Heap?

(Login to Rate and Mark)

3.6

Average Difficulty : 3.6/5.0
Based on 88 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Iterative Quick Sort

Following is a typical recursive implementation of [Quick Sort](#) that uses last element as pivot.

```
/* A typical recursive C/C++ implementation of 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 */
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted,
   l --> Starting index,
   h --> Ending index */
void quickSort(int A[], int l, int h)
{
    if (l < h)
    {
        /* Partitioning index */
        int p = partition(A, l, h);
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }
}
```

[Run on IDE](#)

Python

```
# A typical recursive Python implementation of 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
def partition(arr,low,high):
    i = ( low-1 )           # index of smaller element
    pivot = arr[high]        # pivot

    for j in range(low , high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# This code is contributed by Mohit Kumra
```

[Run on IDE](#)

Java

```
// Java program for implementation of QuickSort
class 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 */
    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-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 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 */
void qSort(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
        qSort(arr, low, pi-1);
        qSort(arr, pi+1, high);
    }
}

```

[Run on IDE](#)

The above implementation can be optimized in many ways

- 1) The above implementation uses last index as pivot. This causes worst-case behavior on already sorted arrays, which is a commonly occurring case. The problem can be solved by choosing either a random index for the pivot, or choosing the middle index of the partition or choosing the median of the first, middle and last element of the partition for the pivot. (See [this](#) for details)
- 2) To reduce the recursion depth, recur first for the smaller half of the array, and use a tail call to recurse into the other.
- 3) Insertion sort works better for small subarrays. Insertion sort can be used for invocations on such small arrays (i.e. where the length is less than a threshold t determined experimentally). For example, [this](#) library implementation of qsort uses insertion sort below size 7.

Despite above optimizations, the function remains recursive and uses [function call stack](#) to store intermediate values of l and h. The function call stack stores other bookkeeping information together with parameters. Also, function calls involve overheads like storing activation record of the caller function and then resuming execution.

The above function can be easily converted to iterative version with the help of an auxiliary stack. Following is an iterative implementation of the above recursive code.

```

// An iterative implementation of quick sort
#include <stdio.h>

// A utility function to swap two elements
void swap ( int* a, int* b )
{
    int t = *a;
    *a = *b;
    *b = t;
}

```

```

        *b = t;
    }

/* This function is same in both iterative and recursive*/
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted,
   l --> Starting index,
   h --> Ending index */
void quickSortIterative (int arr[], int l, int h)
{
    // Create an auxiliary stack
    int stack[ h - l + 1 ];

    // initialize top of stack
    int top = -1;

    // push initial values of l and h to stack
    stack[ ++top ] = l;
    stack[ ++top ] = h;

    // Keep popping from stack while is not empty
    while ( top >= 0 )
    {
        // Pop h and l
        h = stack[ top-- ];
        l = stack[ top-- ];

        // Set pivot element at its correct position
        // in sorted array
        int p = partition( arr, l, h );

        // If there are elements on left side of pivot,
        // then push left side to stack
        if ( p-1 > l )
        {
            stack[ ++top ] = l;
            stack[ ++top ] = p - 1;
        }

        // If there are elements on right side of pivot,
        // then push right side to stack
        if ( p+1 < h )
        {
            stack[ ++top ] = p + 1;
            stack[ ++top ] = h;
        }
    }
}

// A utility function to print contents of arr
void printArr( int arr[], int n )
{
    int i;
    for ( i = 0; i < n; ++i )
        printf( "%d ", arr[i] );
}

// Driver program to test above functions

```

```

int main()
{
    int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};
    int n = sizeof( arr ) / sizeof( *arr );
    quickSortIterative( arr, 0, n - 1 );
    printArr( arr, n );
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program for implementation of Quicksort

# This function is same in both iterative and recursive
def partition(arr,l,h):
    i = ( l - 1 )
    x = arr[h]

    for j in range(l , h):
        if arr[j] <= x:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[h] = arr[h],arr[i+1]
    return (i+1)

# Function to do Quick sort
# arr[] --> Array to be sorted,
# l --> Starting index,
# h --> Ending index
def quickSortIterative(arr,l,h):

    # Create an auxiliary stack
    size = h - l + 1
    stack = [0] * (size)

    # initialize top of stack
    top = -1

    # push initial values of l and h to stack
    top = top + 1
    stack[top] = l
    top = top + 1
    stack[top] = h

    # Keep popping from stack while is not empty
    while top >= 0:

        # Pop h and l
        h = stack[top]
        top = top - 1
        l = stack[top]
        top = top - 1

        # Set pivot element at its correct position in
        # sorted array
        p = partition( arr, l, h )

        # If there are elements on left side of pivot,
        # then push left side to stack
        if p-1 > l:
            top = top + 1
            stack[top] = l
            top = top + 1
            stack[top] = p - 1

        # If there are elements on right side of pivot,

```

```

# then push right side to stack
if p+1 < h:
    top = top + 1
    stack[top] = p + 1
    top = top + 1
    stack[top] = h

# Driver code to test above
arr = [4, 3, 5, 2, 1, 3, 2, 3]
n = len(arr)
quickSortIterative(arr, 0, n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra

```

[Run on IDE](#)

Java

```

// Java implementation of iterative quick sort
class IterativeQuickSort
{
    void swap(int arr[],int i,int j)
    {
        int t = arr[i];
        arr[i] = arr[j];
        arr[j] = t;
    }

    /* This function is same in both iterative and
       recursive*/
    int partition (int arr[], int l, int h)
    {
        int x = arr[h];
        int i = (l - 1);

        for (int j = l; j <= h- 1; j++)
        {
            if (arr[j] <= x)
            {
                i++;
                // swap arr[i] and arr[j]
                swap(arr,i,j);
            }
        }
        // swap arr[i+1] and arr[h]
        swap(arr,i+1,h);
        return (i + 1);
    }

    // Sorts arr[l..h] using iterative QuickSort
    void QuickSort(int arr[], int l, int h)
    {
        // create auxiliary stack
        int stack[] = new int[h-l+1];

        // initialize top of stack
        int top = -1;

        // push initial values in the stack
        stack[++top] = l;
        stack[++top] = h;

        // keep popping elements until stack is not empty
        while (top >= 0)
        {
            // pop h and l
            h = stack[top--];
            l = stack[top--];

```

```

// set pivot element at it's proper position
int p = partition(arr, l, h);

// If there are elements on left side of pivot,
// then push left side to stack
if ( p-1 > l )
{
    stack[ ++top ] = l;
    stack[ ++top ] = p - 1;
}

// If there are elements on right side of pivot,
// then push right side to stack
if ( p+1 < h )
{
    stack[ ++top ] = p + 1;
    stack[ ++top ] = h;
}
}

// A utility function to print contents of arr
void printArr( int arr[], int n )
{
    int i;
    for ( i = 0; i < n; ++i )
        System.out.print(arr[i]+ " ");
}

// Driver code to test above
public static void main(String args[])
{
    IterativeQuickSort ob = new IterativeQuickSort();
    int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};
    ob.QuickSort(arr, 0, arr.length-1);
    ob.printArr(arr, arr.length);
}
}
/*This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

1 2 2 3 3 3 4 5

The above mentioned optimizations for recursive quick sort can also be applied to iterative version.

- 1) Partition process is same in both recursive and iterative. The same techniques to choose optimal pivot can also be applied to iterative version.
- 2) To reduce the stack size, first push the indexes of smaller half.
- 3) Use insertion sort when the size reduces below a experimentally calculated threshold.

References:

<http://en.wikipedia.org/wiki/Quicksort>

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Sorting Quick Sort

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

3.6

Average Difficulty : 3.6/5.0
Based on 27 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

QuickSort on Singly Linked List

QuickSort on Doubly Linked List is discussed [here](#). QuickSort on Singly linked list was given as an exercise.

Following is C++ implementation for same. The important things about implementation are, it changes pointers rather swapping data and time complexity is same as the implementation for Doubly Linked List.

In **partition()**, we consider last element as pivot. We traverse through the current list and if a node has value greater than pivot, we move it after tail. If the node has smaller value, we keep it at its current position.

In **QuickSortRecur()**, we first call partition() which places pivot at correct position and returns pivot. After pivot is placed at correct position, we find tail node of left side (list before pivot) and recur for left list. Finally, we recur for right list.

```
// C++ program for Quick Sort on Singly Linled List
#include <iostream>
#include <cstring>
using namespace std;

/* a node of the singly linked list */
struct node
{
    int data;
    struct node *next;
};

/* A utility function to insert a node at the beginning of linked list */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node = new 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 linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

// Returns the last node of the list
struct node *getTail(struct node *curr)
{
```

```

while (cur != NULL && cur->next != NULL)
    cur = cur->next;
return cur;
}

// Partitions the list taking the last element as the pivot
struct node *partition(struct node *head, struct node *end,
                      struct node **newHead, struct node **newEnd)
{
    struct node *pivot = end;
    struct node *prev = NULL, *cur = head, *tail = pivot;

    // During partition, both the head and end of the list might change
    // which is updated in the newHead and newEnd variables
    while (cur != pivot)
    {
        if (cur->data < pivot->data)
        {
            // First node that has a value less than the pivot - becomes
            // the new head
            if ((*newHead) == NULL)
                (*newHead) = cur;

            prev = cur;
            cur = cur->next;
        }
        else // If cur node is greater than pivot
        {
            // Move cur node to next of tail, and change tail
            if (prev)
                prev->next = cur->next;
            struct node *tmp = cur->next;
            cur->next = NULL;
            tail->next = cur;
            tail = cur;
            cur = tmp;
        }
    }

    // If the pivot data is the smallest element in the current list,
    // pivot becomes the head
    if ((*newHead) == NULL)
        (*newHead) = pivot;

    // Update newEnd to the current last node
    (*newEnd) = tail;

    // Return the pivot node
    return pivot;
}

//here the sorting happens exclusive of the end node
struct node *quickSortRecur(struct node *head, struct node *end)
{
    // base condition
    if (!head || head == end)
        return head;

    node *newHead = NULL, *newEnd = NULL;

    // Partition the list, newHead and newEnd will be updated
    // by the partition function
    struct node *pivot = partition(head, end, &newHead, &newEnd);

    // If pivot is the smallest element - no need to recur for
    // the left part.
    if (newHead != pivot)
    {
        // Set the node before the pivot node as NULL
        struct node *tmp = newHead;
        while (tmp->next != pivot)
            tmp = tmp->next;
        tmp->next = NULL;
    }
}

```

```

// Recur for the list before pivot
newHead = quickSortRecur(newHead, tmp);

// Change next of last node of the left half to pivot
tmp = getTail(newHead);
tmp->next = pivot;
}

// Recur for the list after the pivot element
pivot->next = quickSortRecur(pivot->next, newEnd);

return newHead;
}

// The main function for quick sort. This is a wrapper over recursive
// function quickSortRecur()
void quickSort(struct node **headRef)
{
    (*headRef) = quickSortRecur(*headRef, getTail(*headRef));
    return;
}

// Driver program to test above functions
int main()
{
    struct node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    cout << "Linked List before sorting \n";
    printList(a);

    quickSort(&a);

    cout << "Linked List after sorting \n";
    printList(a);

    return 0;
}

```

[Run on IDE](#)

Output:

```

Linked List before sorting
30 3 4 20 5
Linked List after sorting
3 4 5 20 30

```

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



Take Harvard's
most popular computer
course *online*



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Linked Lists](#) [Sorting](#) [Quick Sort](#)

Related Posts:

- Iteratively Reverse a linked list using only 2 pointers (An Interesting Method)
- Find pair for given sum in a sorted singly linked without extra space
- Find pairs with given sum in doubly linked list
- Unrolled Linked List | Set 1 (Introduction)
- Convert a Binary Tree to a Circular Doubly Link List
- Subtract Two Numbers represented as Linked Lists
- Rearrange a given list such that it consists of alternating minimum maximum elements
- Flatten a multi-level linked list | Set 2 (Depth wise)

(Login to Rate and Mark)

4.5

Average Difficulty : 4.5/5.0
Based on 56 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

QuickSort on Doubly Linked List

Following is a typical recursive implementation of [QuickSort](#) for arrays. The implementation uses last element as pivot.

```
/* A typical recursive implementation of Quicksort for array*/
/* 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 */
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

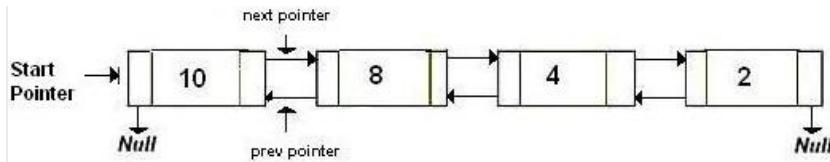
    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted, l --> Starting index, h --> Ending index */
void quickSort(int A[], int l, int h)
{
    if (l < h)
    {
        int p = partition(A, l, h); /* Partitioning index */
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }
}
```

[Run on IDE](#)

Can we use same algorithm for Linked List?

Following is C++ implementation for doubly linked list. The idea is simple, we first find out pointer to last node. Once we have pointer to last node, we can recursively sort the linked list using pointers to first and last nodes of linked list, similar to the above recursive function where we pass indexes of first and last array elements. The partition function for linked list is also similar to partition for arrays. Instead of returning index of the pivot element, it returns pointer to the pivot element. In the following implementation, quickSort() is just a wrapper function, the main recursive function is _quickSort() which is similar to quickSort() for array implementation.



```

// A C++ program to sort a linked list using Quicksort
#include <iostream>
#include <stdio.h>
using namespace std;

/* a node of the doubly linked list */
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};

/* A utility function to swap two elements */
void swap ( int* a, int* b )
{   int t = *a;      *a = *b;      *b = t;  }

// A utility function to find last node of linked list
struct node *lastNode(node *root)
{
    while (root && root->next)
        root = root->next;
    return root;
}

/* Considers 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 */
node* partition(node *l, node *h)
{
    // set pivot as h element
    int x = h->data;

    // similar to i = l-1 for array implementation
    node *i = l->prev;

    // Similar to "for (int j = l; j <= h- 1; j++)"
    for (node *j = l; j != h; j = j->next)
    {
        if (j->data <= x)
        {
            // Similar to i++ for array
            i = (i == NULL)? l : i->next;

            swap(&(i->data), &(j->data));
        }
    }
    i = (i == NULL)? l : i->next; // Similar to i++
    swap(&(i->data), &(h->data));
    return i;
}

/* A recursive implementation of quicksort for linked list */
void _quickSort(struct node* l, struct node *h)
{
    if (h != NULL && l != h && l != h->next)
    {
        struct node *p = partition(l, h);
        _quickSort(l, p->prev);
        _quickSort(p->next, h);
    }
}
  
```

```

// The main function to sort a linked list. It mainly calls _quickSort()
void quickSort(struct node *head)
{
    // Find last node
    struct node *h = lastNode(head);

    // Call the recursive QuickSort
    _quickSort(head, h);
}

// A utility function to print contents of arr
void printList(struct node *head)
{
    while (head)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

/* Function to insert a node at the begining of the Doubly Linked List */
void push(struct node** head_ref, int new_data)
{
    struct node* new_node = new node;      /* allocate node */
    new_node->data = new_data;

    /* since we are adding at the begining, prev is always NULL */
    new_node->prev = NULL;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* change prev of head node to new node */
    if ((*head_ref) != NULL) (*head_ref)->prev = new_node ;

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Driver program to test above function */
int main()
{
    struct node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    cout << "Linked List before sorting \n";
    printList(a);

    quickSort(a);

    cout << "Linked List after sorting \n";
    printList(a);

    return 0;
}

```

[Run on IDE](#)

Java

```

// A Java program to sort a linked list using Quicksort
class QuickSort_using_Doubly_LinkedList{
    Node head;

    /* a node of the doubly linked list */

```

```

static class Node{
    private int data;
    private Node next;
    private Node prev;

    Node(int d){
        data = d;
        next = null;
        prev = null;
    }
}

// A utility function to find last node of linked list
Node lastNode(Node node){
    while(node.next!=null)
        node = node.next;
    return node;
}

/* Considers 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 */
Node partition(Node l,Node h)
{
    // set pivot as h element
    int x = h.data;

    // similar to i = l-1 for array implementation
    Node i = l.prev;

    // Similar to "for (int j = l; j <= h- 1; j++)"
    for(Node j=l; j!=h; j=j.next)
    {
        if(j.data <= x)
        {
            // Similar to i++ for array
            i = (i==null) ? l : i.next;
            int temp = i.data;
            i.data = j.data;
            j.data = temp;
        }
    }
    i = (i==null) ? l : i.next; // Similar to i++
    int temp = i.data;
    i.data = h.data;
    h.data = temp;
    return i;
}

/* A recursive implementation of quicksort for linked list */
void _quickSort(Node l,Node h)
{
    if(h!=null && l!=h && l!=h.next){
        Node temp = partition(l,h);
        _quickSort(l,temp.prev);
        _quickSort(temp.next,h);
    }
}

// The main function to sort a linked list. It mainly calls _quickSort()
public void quickSort(Node node)
{
    // Find last node
    Node head = lastNode(node);

    // Call the recursive QuickSort
    _quickSort(node,head);
}

// A utility function to print contents of arr
public void printList(Node head)
{
    while(head!=null){

```

```

        System.out.print(head.data+" ");
        head = head.next;
    }

/* Function to insert a node at the begining of the Doubly Linked List */
void push(int new_Data)
{
    Node new_Node = new Node(new_Data);      /* allocate node */

    // if head is null, head = new_Node
    if(head==null){
        head = new_Node;
        return;
    }

    /* link the old list off the new node */
    new_Node.next = head;

    /* change prev of head node to new node */
    head.prev = new_Node;

    /* since we are adding at the begining, prev is always NULL */
    new_Node.prev = null;

    /* move the head to point to the new node */
    head = new_Node;
}

/* Driver program to test above function */
public static void main(String[] args){
    QuickSort_using_Doubly_LinkedList list = new QuickSort_using_Doubly_LinkedList();

    list.push(5);
    list.push(20);
    list.push(4);
    list.push(3);
    list.push(30);

    System.out.println("Linked List before sorting ");
    list.printList(list.head);
    System.out.println("\nLinked List after sorting");
    list.quickSort(list.head);
    list.printList(list.head);
}
}

// This code has been contributed by Amit Khandelwal

```

[Run on IDE](#)

Output :

```

Linked List before sorting
30 3 4 20 5
Linked List after sorting
3 4 5 20 30

```

Time Complexity: Time complexity of the above implementation is same as time complexity of QuickSort() for arrays. It takes $O(n^2)$ time in worst case and $O(n\log n)$ in average and best cases. The worst case occurs when the linked list is already sorted.

Can we implement random quick sort for linked list?

Quicksort can be implemented for Linked List only when we can pick a fixed point as pivot (like last element in above implementation). Random QuickSort cannot be efficiently implemented for Linked Lists by picking random pivot.

Exercise:

The above implementation is for doubly linked list. Modify it for singly linked list. Note that we don't have prev pointer in singly linked list.

Refer [QuickSort on Singly Linked List](#) for solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Linked Lists](#) [Sorting](#) [Quick Sort](#)

Related Posts:

- Iteratively Reverse a linked list using only 2 pointers (An Interesting Method)
- Find pair for given sum in a sorted singly linked without extra space
- Find pairs with given sum in doubly linked list
- Unrolled Linked List | Set 1 (Introduction)
- Convert a Binary Tree to a Circular Doubly Link List
- Subtract Two Numbers represented as Linked Lists
- Rearrange a given list such that it consists of alternating minimum maximum elements
- Flatten a multi-level linked list | Set 2 (Depth wise)

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 40 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find k closest elements to a given value

Given a sorted array arr[] and a value X, find the k closest elements to X in arr[].

Examples:

```
Input: K = 4, X = 35
       arr[] = {12, 16, 22, 30, 35, 39, 42,
                 45, 48, 50, 53, 55, 56}
Output: 30 39 42 45
```

Note that if the element is present in array, then it should not be in output, only the other closest elements are required.

In the following solutions, it is assumed that all elements of array are distinct.

A **simple solution** is to do linear search for k closest elements.

- 1) Start from the first element and search for the crossover point (The point before which elements are smaller than or equal to X and after which elements are greater). This step takes O(n) time.
- 2) Once we find the crossover point, we can compare elements on both sides of crossover point to print k closest elements. This step takes O(k) time.

The time complexity of the above solution is O(n).

An **Optimized Solution** is to find k elements in O(Logn + k) time. The idea is to use [Binary Search](#) to find the crossover point. Once we find index of crossover point, we can print k closest elements in O(k) time.

```
#include<stdio.h>

/* Function to find the cross over point (the point before
   which elements are smaller than or equal to x and after
   which greater than x)*/
int findCrossOver(int arr[], int low, int high, int x)
{
    // Base cases
    if (arr[high] <= x) // x is greater than all
        return high;
    if (arr[low] > x) // x is smaller than all
        return low;

    // Find the middle point
    int mid = (low + high)/2; /* low + (high - low)/2 */

    /* If x is same as middle element, then return mid */
    if (arr[mid] <= x && arr[mid+1] > x)
        return mid;
```

```

/* If x is greater than arr[mid], then either arr[mid + 1]
   is ceiling of x or ceiling lies in arr[mid+1...high] */
if(arr[mid] < x)
    return findCrossOver(arr, mid+1, high, x);

return findCrossOver(arr, low, mid - 1, x);
}

// This function prints k closest elements to x in arr[].
// n is the number of elements in arr[]
void printKclosest(int arr[], int x, int k, int n)
{
    // Find the crossover point
    int l = findCrossOver(arr, 0, n-1, x);
    int r = l+1;    // Right index to search
    int count = 0; // To keep track of count of elements already printed

    // If x is present in arr[], then reduce left index
    // Assumption: all elements in arr[] are distinct
    if (arr[l] == x) l--;

    // Compare elements on left and right of crossover
    // point to find the k closest elements
    while (l >= 0 && r < n && count < k)
    {
        if (x - arr[l] < arr[r] - x)
            printf("%d ", arr[l--]);
        else
            printf("%d ", arr[r++]);
        count++;
    }

    // If there are no more elements on right side, then
    // print left elements
    while (count < k && l >= 0)
        printf("%d ", arr[l--]), count++;

    // If there are no more elements on left side, then
    // print right elements
    while (count < k && r < n)
        printf("%d ", arr[r++]), count++;
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {12, 16, 22, 30, 35, 39, 42,
                  45, 48, 50, 53, 55, 56};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 35, k = 4;
    printKclosest(arr, x, 4, n);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to find k closest elements to a given value
class KClosest
{
    /* Function to find the cross over point (the point before
       which elements are smaller than or equal to x and after
       which greater than x)*/
    int findCrossOver(int arr[], int low, int high, int x)
    {
        // Base cases
        if (arr[high] <= x) // x is greater than all
            return high;
        if (arr[low] > x) // x is smaller than all

```

```

        return low;

    // Find the middle point
    int mid = (low + high)/2; /* low + (high - low)/2 */

    /* If x is same as middle element, then return mid */
    if (arr[mid] <= x && arr[mid+1] > x)
        return mid;

    /* If x is greater than arr[mid], then either arr[mid + 1]
       is ceiling of x or ceiling lies in arr[mid+1...high] */
    if(arr[mid] < x)
        return findCrossOver(arr, mid+1, high, x);

    return findCrossOver(arr, low, mid - 1, x);
}

// This function prints k closest elements to x in arr[].
// n is the number of elements in arr[]
void printKclosest(int arr[], int x, int k, int n)
{
    // Find the crossover point
    int l = findCrossOver(arr, 0, n-1, x);
    int r = l+1; // Right index to search
    int count = 0; // To keep track of count of elements
                   // already printed

    // If x is present in arr[], then reduce left index
    // Assumption: all elements in arr[] are distinct
    if (arr[l] == x) l--;

    // Compare elements on left and right of crossover
    // point to find the k closest elements
    while (l >= 0 && r < n && count < k)
    {
        if (x - arr[l] < arr[r] - x)
            System.out.print(arr[l--]+ " ");
        else
            System.out.print(arr[r++]+ " ");
        count++;
    }

    // If there are no more elements on right side, then
    // print left elements
    while (count < k && l >= 0)
    {
        System.out.print(arr[l--]+ " ");
        count++;
    }

    // If there are no more elements on left side, then
    // print right elements
    while (count < k && r < n)
    {
        System.out.print(arr[r++]+ " ");
        count++;
    }
}

/* Driver program to check above functions */
public static void main(String args[])
{
    KClosest ob = new KCclosest();
    int arr[] = {12, 16, 22, 30, 35, 39, 42,
                45, 48, 50, 53, 55, 56
               };
    int n = arr.length;
    int x = 35, k = 4;
    ob.printKclosest(arr, x, 4, n);
}
*/
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Output:

```
39 30 42 45
```

The time complexity of this method is $O(\log n + k)$.

Exercise: Extend the optimized solution to work for duplicates also, i.e., to work for arrays where elements don't have to be distinct.

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



Download

Zip, Unzip or Open Any File
With Unzipper.

Unzipper



GATE CS Corner Company Wise Coding Practice

Searching

Related Posts:

- Second minimum element using minimum comparisons
- [Interpolation Search](#)
- [Jump Search](#)
- Third largest element in an array of distinct elements
- [Floor in a Sorted Array](#)
- [Fibonacci Search](#)
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

[\(Login to Rate and Mark\)](#)**3.3**Average Difficulty : **3.3/5.0**
Based on **34** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Sort n numbers in range from 0 to $n^2 - 1$ in linear time

Given an array of numbers of size n. It is also given that the array elements are in range from 0 to $n^2 - 1$. Sort the given array in linear time.

Examples:

Since there are 5 elements, the elements can be from 0 to 24.

Input: arr[] = {0, 23, 14, 12, 9}

Output: arr[] = {0, 9, 12, 14, 23}

Since there are 3 elements, the elements can be from 0 to 8.

Input: arr[] = {7, 0, 2}

Output: arr[] = {0, 2, 7}

We strongly recommend to minimize the browser and try this yourself first.

Solution: If we use [Counting Sort](#), it would take $O(n^2)$ time as the given range is of size n^2 . Using any comparison based sorting like [Merge Sort](#), [Heap Sort](#), .. etc would take $O(n \log n)$ time.

Now question arises how to do this in $O(n)$? Firstly, is it possible? Can we use data given in question? n numbers in range from 0 to $n^2 - 1$?

The idea is to use [Radix Sort](#). Following is standard Radix Sort algorithm.

- 1) Do following for each digit i where i varies from least significant digit to the most significant digit.
 - a) Sort input array using counting sort (or any stable sort) according to the i'th digit

Let there be d digits in input integers. Radix Sort takes $O(d*(n+b))$ time where b is the base for representing numbers, for example, for decimal system, b is 10. Since n^2-1 is the maximum possible value, the value of d would be $O(\log_b(n))$. So overall time complexity is $O((n+b)*O(\log_b(n)))$. Which looks more than the time complexity of comparison based sorting algorithms for a large k. The idea is to change base b. If we set b as n, the value of $O(\log_b(n))$ becomes $O(1)$ and overall time complexity becomes $O(n)$.

arr[] = {0, 10, 13, 12, 7}

Let us consider the elements in base 5. For example 13 in base 5 is 23, and 7 in base 5 is 12.

arr[] = {00(0), 20(10), 23(13), 22(12), 12(7)}

After first iteration (Sorting according to the last digit in

```
base 5), we get.
```

```
arr[] = {00(0), 20(10), 12(7), 22(12), 23(13)}
```

After second iteration, we get

```
arr[] = {00(0), 12(7), 20(10), 22(12), 23(13)}
```

Following is C++ implementation to sort an array of size n where elements are in range from 0 to $n^2 - 1$.

```
#include<iostream>
using namespace std;

// A function to do counting sort of arr[] according to
// the digit represented by exp.
int countSort(int arr[], int n, int exp)
{
    int output[n]; // output array
    int i, count[n];
    for (int i=0; i < n; i++)
        count[i] = 0;

    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
        count[(arr[i]/exp)%n]++;
    
    // Change count[i] so that count[i] now contains actual
    // position of this digit in output[]
    for (i = 1; i < n; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i]/exp)%n] - 1] = arr[i];
        count[(arr[i]/exp)%n]--;
    }

    // Copy the output array to arr[], so that arr[] now
    // contains sorted numbers according to current digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using Radix Sort
void sort(int arr[], int n)
{
    // Do counting sort for first digit in base n. Note that
    // instead of passing digit number, exp (n^0 = 0) is passed.
    countSort(arr, n, 1);

    // Do counting sort for second digit in base n. Note that
    // instead of passing digit number, exp (n^1 = n) is passed.
    countSort(arr, n, n);
}

// A utility function to print an array
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above functions
int main()
{
    // Since array size is 7, elements should be from 0 to 48
    int arr[] = {40, 12, 45, 32, 33, 1, 22};
```

```

int n = sizeof(arr)/sizeof(arr[0]);
cout << "Given array is \n";
printArr(arr, n);

sort(arr, n);

cout << "\nSorted array is \n";
printArr(arr, n);
return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to sort an array of size n where elements are
// in range from 0 to  $n^2 - 1$ .
class Sort1ToN2
{
    // A function to do counting sort of arr[] according to
    // the digit represented by exp.
    void countSort(int arr[], int n, int exp)
    {
        int output[] = new int[n]; // output array
        int i, count[] = new int[n] ;
        for (i=0; i < n; i++)
            count[i] = 0;

        // Store count of occurrences in count[]
        for (i = 0; i < n; i++)
            count[ (arr[i]/exp)%n ]++;

        // Change count[i] so that count[i] now contains actual
        // position of this digit in output[]
        for (i = 1; i < n; i++)
            count[i] += count[i - 1];

        // Build the output array
        for (i = n - 1; i >= 0; i--)
        {
            output[count[ (arr[i]/exp)%n ] - 1] = arr[i];
            count[(arr[i]/exp)%n]--;
        }

        // Copy the output array to arr[], so that arr[] now
        // contains sorted numbers according to current digit
        for (i = 0; i < n; i++)
            arr[i] = output[i];
    }

    // The main function to that sorts arr[] of size n using Radix Sort
    void sort(int arr[], int n)
    {
        // Do counting sort for first digit in base n. Note that
        // instead of passing digit number, exp ( $n^0 = 1$ ) is passed.
        countSort(arr, n, 1);

        // Do counting sort for second digit in base n. Note that
        // instead of passing digit number, exp ( $n^1 = n$ ) is passed.
        countSort(arr, n, n);
    }

    // A utility function to print an array
    void printArr(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i]+ " ");
    }

    // Driver program to test above functions
}

```

```

public static void main(String args[])
{
    Sort1ToN2 ob = new Sort1ToN2();

    // Since array size is 7, elements should be from 0 to 48
    int arr[] = {40, 12, 45, 32, 33, 1, 22};
    int n = arr.length;
    System.out.println("Given array");
    ob.printArr(arr, n);

    ob.sort(arr, n);

    System.out.println("Sorted array");
    ob.printArr(arr, n);
}
/*This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

```

Given array is
40 12 45 32 33 1 22
Sorted array is
1 12 22 32 33 40 45

```

How to sort if range is from 1 to n^2 ?

If range is from 1 to n^2 , the above process can not be directly applied, it must be changed. Consider $n = 100$ and range from 1 to 10000. Since the base is 100, a digit must be from 0 to 99 and there should be 2 digits in the numbers. But the number 10000 has more than 2 digits. So to sort numbers in a range from 1 to n^2 , we can use following process.

- 1) Subtract all numbers by 1.
- 2) Since the range is now 0 to n^2 , do counting sort twice as done in the above implementation.
- 3) After the elements are sorted, add 1 to all numbers to obtain the original numbers.

How to sort if range is from 0 to $n^3 - 1$?

Since there can be 3 digits in base n , we need to call counting sort 3 times.

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



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Sorting

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 14 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

A Problem in Many Binary Search Implementations

Consider the following C implementation of [Binary Search](#) function, is there anything wrong in this?

```
// A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        // find index of middle element
        int m = (l+r)/2;

        // Check if x is present at mid
        if (arr[m] == x) return m;

        // If x greater, ignore left half
        if (arr[m] < x) l = m + 1;

        // If x is smaller, ignore right half
        else r = m - 1;
    }

    // if we reach here, then element was not present
    return -1;
}
```

[Run on IDE](#)

The above looks fine except one subtle thing, the expression “ $m = (l+r)/2$ ”. It fails for large values of l and r . Specifically, it fails if the sum of low and high is greater than the maximum positive int value ($2^{31} - 1$). The sum overflows to a negative value, and the value stays negative when divided by two. In C this causes an array index out of bounds with unpredictable results.

What is the way to resolve this problem?

Following is one way:

```
int mid = low + ((high - low) / 2);
```

Probably faster, and arguably as clear is (works only in Java, refer [this](#)):

```
int mid = (low + high) >>> 1;
```

In C and C++ (where you don't have the $>>>$ operator), you can do this:

```
mid = ((unsigned int)low + (unsigned int)high)) >> 1
```

The similar problem appears in [Merge Sort](#) as well.

The above content is taken from [google reasearch blog](#).

Please refer [this](#) as well, it points out that the above solutions may not always work.

The above problem occurs when array length is 2^{30} or greater and the search repeatedly moves to second half of the array. This much size of array is not likely to appear most of the time. For example, when we try the below program with 32 bit [Code Blocks](#) compiler, we get compiler error.

```
int main()
{
    int arr[1<<30];
    return 0;
}
```

[Run on IDE](#)

Output:

```
error: size of array 'arr' is too large
```

Even when we try boolean array, the program compiles fine, but crashes when run in Windows 7.0 and [Code Blocks](#) 32 bit compiler

```
#include <stdbool.h>
int main()
{
    bool arr[1<<30];
    return 0;
}
```

[Run on IDE](#)

Output: No compiler error, but crashes at run time.

Sources:

<http://googleresearch.blogspot.in/2006/06/extr-extra-read-all-about-it-nearly.html>

http://locklessinc.com/articles/binary_search/

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



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Searching](#) [Binary-Search](#)

Related Posts:

- Second minimum element using minimum comparisons
- [Interpolation Search](#)
- [Jump Search](#)
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

(Login to Rate and Mark)

2.2

Average Difficulty : 2.2/5.0
Based on 28 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Search in an almost sorted array

Given an array which is sorted, but after sorting some elements are moved to either of the adjacent positions, i.e., arr[i] may be present at arr[i+1] or arr[i-1]. Write an efficient function to search an element in this array. Basically the element arr[i] can only be swapped with either arr[i+1] or arr[i-1].

For example consider the array {2, 3, 10, 4, 40}, 4 is moved to next position and 10 is moved to previous position.

Example:

```
Input: arr[] = {10, 3, 40, 20, 50, 80, 70}, key = 40
Output: 2
Output is index of 40 in given array
```

```
Input: arr[] = {10, 3, 40, 20, 50, 80, 70}, key = 90
Output: -1
-1 is returned to indicate element is not present
```

A simple solution is to linearly search the given key in given array. Time complexity of this solution is O(n). We can modify [binary search](#) to do it in O(Logn) time.

The idea is to compare the key with middle 3 elements, if present then return the index. If not present, then compare the key with middle element to decide whether to go in left half or right half. Comparing with middle element is enough as all the elements after mid+2 must be greater than element mid and all elements before mid-2 must be smaller than mid element.

Following is C++ implementation of this approach.

```
// C++ program to find an element in an almost sorted
// array
#include <stdio.h>

// A recursive binary search based function. It returns
// index of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= 1)
    {
        int mid = l + (r - 1)/2;

        // If the element is present at one of the middle
        // 3 positions
        if (arr[mid] == x)  return mid;
        if (mid > 1 && arr[mid-1] == x) return (mid - 1);
    }
}
```

```

        if (mid < r && arr[mid+1] == x) return (mid + 1);

        // If element is smaller than mid, then it can only
        // be present in left subarray
        if (arr[mid] > x) return binarySearch(arr, 1, mid-2, x);

        // Else the element can only be present in right subarray
        return binarySearch(arr, mid+2, r, x);
    }

    // We reach here when element is not present in array
    return -1;
}

// Driver program to test above function
int main(void)
{
    int arr[] = {3, 2, 10, 4, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 4;
    int result = binarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
                  : printf("Element is present at index %d", result);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to find an element in an almost sorted array
class SearchAlmost
{
    // A recursive binary search based function. It returns
    // index of x in given array arr[l..r] is present,
    // otherwise -1
    int binarySearch(int arr[], int l, int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - 1)/2;

            // If the element is present at one of the middle
            // 3 positions
            if (arr[mid] == x) return mid;
            if (mid > 1 && arr[mid-1] == x) return (mid - 1);
            if (mid < r && arr[mid+1] == x) return (mid + 1);

            // If element is smaller than mid, then it can
            // only be present in left subarray
            if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

            // Else the element can only be present in right
            // subarray
            return binarySearch(arr, mid+1, r, x);
        }

        // We reach here when element is not present in array
        return -1;
    }

    // Driver method
    public static void main(String args[])
    {
        abc ob = new abc();
        int arr[] = {3, 2, 10, 4, 40};
        int n = arr.length;
        int x = 4;
        int result = ob.binarySearch(arr, 0, n-1, x);
        if(result == -1)
            System.out.println("Element is not present in array");
    }
}

```

```

        else
            System.out.println("Element is present at index " +
                               result);
    }
/*This code is contributed by Rajat Mishra*/

```

[Run on IDE](#)

Output:

Element is present at index 3

Time complexity of the above function is O(Logn).

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



Download

Zip, Unzip or Open Any File
With Unzipper.

Unzipper



GATE CS Corner Company Wise Coding Practice

Searching

Related Posts:

- Second minimum element using minimum comparisons
- [Interpolation Search](#)
- [Jump Search](#)
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

[\(Login to Rate and Mark\)](#)**2.7**Average Difficulty : **2.7/5.0**
Based on **29** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Sort an array in wave form

Given an unsorted array of integers, sort the array into a wave like array. An array ‘arr[0..n-1]’ is sorted in wave form if $\text{arr}[0] \geq \text{arr}[1] \leq \text{arr}[2] \geq \text{arr}[3] \leq \text{arr}[4] \geq \dots$

Examples:

Input: $\text{arr}[] = \{10, 5, 6, 3, 2, 20, 100, 80\}$
 Output: $\text{arr}[] = \{10, 5, 6, 2, 20, 3, 100, 80\}$ OR
 $\{20, 5, 10, 2, 80, 6, 100, 3\}$ OR
 any other array that is in wave form

Input: $\text{arr}[] = \{20, 10, 8, 6, 4, 2\}$
 Output: $\text{arr}[] = \{20, 8, 10, 4, 6, 2\}$ OR
 $\{10, 8, 20, 2, 6, 4\}$ OR
 any other array that is in wave form

Input: $\text{arr}[] = \{2, 4, 6, 8, 10, 20\}$
 Output: $\text{arr}[] = \{4, 2, 8, 6, 20, 10\}$ OR
 any other array that is in wave form

Input: $\text{arr}[] = \{3, 6, 5, 10, 7, 20\}$
 Output: $\text{arr}[] = \{6, 3, 10, 5, 20, 7\}$ OR
 any other array that is in wave form

We strongly recommend that you click here and practice it, before moving on to the solution.

A **Simple Solution** is to use sorting. First sort the input array, then swap all adjacent elements.

For example, let the input array be $\{3, 6, 5, 10, 7, 20\}$. After sorting, we get $\{3, 5, 6, 7, 10, 20\}$. After swapping adjacent elements, we get $\{5, 3, 7, 6, 20, 10\}$.

Below are implementations of this simple approach.

```
// A C++ program to sort an array in wave form using
// a sorting function
#include<iostream>
```

```
#include<algorithm>
using namespace std;

// A utility method to swap two numbers.
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// This function sorts arr[0..n-1] in wave form, i.e.,
// arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5]..
void sortInWave(int arr[], int n)
{
    // Sort the input array
    sort(arr, arr+n);

    // Swap adjacent elements
    for (int i=0; i<n-1; i += 2)
        swap(&arr[i], &arr[i+1]);
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 90, 49, 2, 1, 5, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortInWave(arr, n);
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

[Run on IDE](#)

Python

```
# Python function to sort the array arr[0..n-1] in wave form,
# i.e., arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5]
def sortInWave(arr, n):

    #sort the array
    arr.sort()

    # Swap adjacent elements
    for i in range(0,n-1,2):
        arr[i], arr[i+1] = arr[i+1], arr[i]

# Driver program
arr = [10, 90, 49, 2, 1, 5, 23]
sortInWave(arr, len(arr))
for i in range(0,len(arr)):
    print arr[i],
```

This code is contributed by __Devesh Agrawal__

[Run on IDE](#)

Java

```
// Java implementation of naive method for sorting
// an array in wave form.
import java.util.*;

class SortWave
{
    // A utility method to swap two numbers.
```

```

void swap(int arr[], int a, int b)
{
    int temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}

// This function sorts arr[0..n-1] in wave form, i.e.,
// arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]..
void sortInWave(int arr[], int n)
{
    // Sort the input array
    Arrays.sort(arr);

    // Swap adjacent elements
    for (int i=0; i<n-1; i += 2)
        swap(arr, i, i+1);
}

// Driver method
public static void main(String args[])
{
    Test ob = new Test();
    int arr[] = {10, 90, 49, 2, 1, 5, 23};
    int n = arr.length;
    ob.sortInWave(arr, n);
    for (int i : arr)
        System.out.print(i + " ");
}
/*This code is contributed by Rajat Mishra*/

```

[Run on IDE](#)

Output:

2 1 10 5 49 23 90

The time complexity of the above solution is $O(n \log n)$ if a $O(n \log n)$ sorting algorithm like **Merge Sort**, **Heap Sort**, .. etc is used.

This can be done in **$O(n)$ time by doing a single traversal** of given array. The idea is based on the fact that if we make sure that all even positioned (at index 0, 2, 4, ..) elements are greater than their adjacent odd elements, we don't need to worry about odd positioned element. Following are simple steps.

1) Traverse all even positioned elements of input array, and do following.

....a) If current element is smaller than previous odd element, swap previous and current.

....b) If current element is smaller than next odd element, swap next and current.

Below are implementations of above simple algorithm.

```

// A O(n) program to sort an input array in wave form
#include<iostream>
using namespace std;

// A utility method to swap two numbers.
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

```

```
// This function sorts arr[0..n-1] in wave form, i.e., arr[0] >=
// arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5] ....
void sortInWave(int arr[], int n)
{
    // Traverse all even elements
    for (int i = 0; i < n; i+=2)
    {
        // If current even element is smaller than previous
        if (i>0 && arr[i-1] > arr[i])
            swap(&arr[i], &arr[i-1]);

        // If current even element is smaller than next
        if (i<n-1 && arr[i] < arr[i+1])
            swap(&arr[i], &arr[i + 1]);
    }
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 90, 49, 2, 1, 5, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortInWave(arr, n);
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

[Run on IDE](#)

Python

```
# Python function to sort the array arr[0..n-1] in wave form,
# i.e., arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5]
def sortInWave(arr, n):

    # Traverse all even elements
    for i in range(0, n, 2):

        # If current even element is smaller than previous
        if (i>0 and arr[i] < arr[i-1]):
            arr[i],arr[i-1] = arr[i-1],arr[i]

        # If current even element is smaller than next
        if (i < n-1 and arr[i] < arr[i+1]):
            arr[i],arr[i+1] = arr[i+1],arr[i]

# Driver program
arr = [10, 90, 49, 2, 1, 5, 23]
sortInWave(arr, len(arr))
for i in range(0,len(arr)):
    print arr[i],
```

This code is contributed by __Devesh Agrawal__

[Run on IDE](#)

Java

```
// A O(n) Java program to sort an input array in wave form
class SortWave
{
    // A utility method to swap two numbers.
    void swap(int arr[], int a, int b)
    {
        int temp = arr[a];
        arr[a] = arr[b];
```

```

        arr[b] = temp;
    }

// This function sorts arr[0..n-1] in wave form, i.e.,
// arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]....
void sortInWave(int arr[], int n)
{
    // Traverse all even elements
    for (int i = 0; i < n; i+=2)
    {
        // If current even element is smaller
        // than previous
        if (i>0 && arr[i-1] > arr[i] )
            swap(arr, i-1, i);

        // If current even element is smaller
        // than next
        if (i<n-1 && arr[i] < arr[i+1] )
            swap(arr, i, i + 1);
    }
}

// Driver program to test above function
public static void main(String args[])
{
    SortWave ob = new SortWave();
    int arr[] = {10, 90, 49, 2, 1, 5, 23};
    int n = arr.length;
    ob.sortInWave(arr, n);
    for (int i : arr)
        System.out.print(i+" ");
}
/*This code is contributed by Rajat Mishra*/

```

[Run on IDE](#)

Output:

90 10 49 1 5 2 23

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



Download

Free Download

www.unzipper.com



[Sorting](#) [array-rearrange](#)

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2.7Average Difficulty : **2.7/5.0**
Based on **51** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Why is Binary Search preferred over Ternary Search?

The following is a simple recursive **Binary Search** function in C++ taken from [here](#).

```
// A recursive binary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= 1)
    {
        int mid = l + (r - 1)/2;

        // If the element is present at the middle itself
        if (arr[mid] == x) return mid;

        // If element is smaller than mid, then it can only be present
        // in left subarray
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present in right subarray
        return binarySearch(arr, mid+1, r, x);
    }

    // We reach here when element is not present in array
    return -1;
}
```

[Run on IDE](#)

The following is a simple recursive **Ternary Search** function in C++.

```
// A recursive ternary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int ternarySearch(int arr[], int l, int r, int x)
{
    if (r >= 1)
    {
        int mid1 = l + (r - 1)/3;
        int mid2 = mid1 + (r - 1)/3;

        // If x is present at the mid1
        if (arr[mid1] == x) return mid1;

        // If x is present at the mid2
        if (arr[mid2] == x) return mid2;

        // If x is present in left one-third
        if (arr[mid1] > x) return ternarySearch(arr, l, mid1-1, x);

        // If x is present in right one-third
        if (arr[mid2] < x) return ternarySearch(arr, mid2+1, r, x);

        // If x is present in middle one-third
        return ternarySearch(arr, mid1+1, mid2-1, x);
    }
}
```

```
// We reach here when element is not present in array
return -1;
}
```

[Run on IDE](#)

Which of the above two does less comparisons in worst case?

From the first look, it seems the ternary search does less number of comparisons as it makes $\log_3 n$ recursive calls, but binary search makes $\log_2 n$ recursive calls. Let us take a closer look.

The following is recursive formula for counting comparisons in worst case of Binary Search.

$$T(n) = T(n/2) + 2, \quad T(1) = 1$$

The following is recursive formula for counting comparisons in worst case of Ternary Search.

$$T(n) = T(n/3) + 4, \quad T(1) = 1$$

In binary search, there are $2\log_2 n + 1$ comparisons in worst case. In ternary search, there are $4\log_3 n + 1$ comparisons in worst case.

Therefore, the comparison of Ternary and Binary Searches boils down the comparison of expressions $2\log_3 n$ and $\log_2 n$. The value of $2\log_3 n$ can be written as $(2 / \log_2 3) * \log_2 n$. Since the value of $(2 / \log_2 3)$ is more than one, Ternary Search does more comparisons than Binary Search in worst case.

Exercise:

Why Merge Sort divides input array in two halves, why not in three or more parts?

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



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Searching

Related Posts:

- Second minimum element using minimum comparisons
- Interpolation Search
- Jump Search
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

(Login to Rate and Mark)

2.8

Average Difficulty : 2.8/5.0
Based on 18 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

Practice

GATE CS

Placements

GeeksQuiz

Google™ Custom Search



Login/Register

K'th Smallest/Largest Element in Unsorted Array | Set 1

Given an array and a number k where k is smaller than size of array, we need to find the k'th smallest element in the given array. It is given that all array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7
```

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

We have discussed a similar [problem](#) to print k largest elements.

Method 1 (Simple Solution)

A Simple Solution is to sort the given array using a O(nlogn) sorting algorithm like [Merge Sort](#), [Heap Sort](#), etc and return the element at index k-1 in the sorted array. Time Complexity of this solution is O(nLogn).

```
// Simple C++ program to find k'th smallest element
#include<iostream>
#include<algorithm>
using namespace std;

// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Sort the given array
    sort(arr, arr+n);

    // Return k'th element in the sorted array
    return arr[k-1];
}

// Driver program to test above methods
int main()
{
```

```

int arr[] = {12, 3, 5, 7, 19};
int n = sizeof(arr)/sizeof(arr[0]), k = 2;
cout << "K'th smallest element is " << kthSmallest(arr, n, k);
return 0;
}

```

K'th smallest element is 5

Method 2 (Using Min Heap – HeapSelect)

We can find k'th smallest element in time complexity better than O(nLogn). A simple optimization is to create a **Min Heap** of the given n elements and call extractMin() k times.

The following is C++ implementation of above method.

```

// A C++ program to find k'th smallest element using min heap
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    MinHeap(int a[], int size); // Constructor
    void MinHeapify(int i); // To minheapify subtree rooted with index i
    int parent(int i) { return (i-1)/2; }
    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }

    int extractMin(); // extracts root (minimum) element
    int getMin() { return harr[0]; } // Returns minimum
};

MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{

```

```

if (heap_size == 0)
    return INT_MAX;

// Store the minimum value.
int root = harr[0];

// If there are more than 1 items, move the last item to root
// and call heapify.
if (heap_size > 1)
{
    harr[0] = harr[heap_size-1];
    MinHeapify(0);
}
heap_size--;

return root;
}

// 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] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Build a heap of n elements: O(n) time
    MinHeap mh(arr, n);

    // Do extract min (k-1) times
    for (int i=0; i<k-1; i++)
        mh.extractMin();

    // Return root
    return mh.getMin();
}

```

```
// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 2;
    cout << "K'th smallest element is " << kthSmallest(arr, n, k);
    return 0;
}
```

Output:

K'th smallest element is 5

Time complexity of this solution is $O(n + k\log n)$.

Method 3 (Using Max-Heap)

We can also use Max Heap for finding the k'th smallest element. Following is algorithm.

- 1) Build a Max-Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. $O(k \log k)$
- 2) For each element, after the k'th element (arr[k] to arr[n-1]), compare it with root of MH.
 -a) If the element is less than the root then make it root and call heapify for MH
 -b) Else ignore it.
- 3) Finally, root of the MH is the kth smallest element.

Time complexity of this solution is $O(k + (n-k)\log k)$

The following is C++ implementation of above algorithm

```
// A C++ program to find k'th smallest element using max heap
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Max Heap
class MaxHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of max heap
    int heap_size; // Current number of elements in max heap
public:
    MaxHeap(int a[], int size); // Constructor
    void maxHeapify(int i); // To maxHeapify subtree rooted with index i
    int parent(int i) { return (i-1)/2; }
    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }

    int extractMax(); // extracts root (maximum) element
    int getMax() { return harr[0]; } // Returns maximum
```

```

// to replace root with new node x and heapify() new root
void replaceMax(int x) { harr[0] = x; maxHeapify(0); }

};

MaxHeap::MaxHeap(int a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        maxHeapify(i);
        i--;
    }
}

// Method to remove maximum element (or root) from max heap
int MaxHeap::extractMax()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the maximum value.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        maxHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MaxHeap::maxHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int largest = i;
    if (l < heap_size && harr[l] > harr[i])
        largest = l;
    if (r < heap_size && harr[r] > harr[largest])
        largest = r;
    if (largest != i)
    {
        swap(&harr[i], &harr[largest]);
        maxHeapify(largest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
}

```

```

int temp = *x;
*x = *y;
*y = temp;
}

// Function to return k'th largest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Build a heap of first k elements: O(k) time
    MaxHeap mh(arr, k);

    // Process remaining n-k elements. If current element is
    // smaller than root, replace root with current element
    for (int i=k; i<n; i++)
        if (arr[i] < mh.getMax())
            mh.replaceMax(arr[i]);

    // Return root
    return mh.getMax();
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 4;
    cout << "K'th smallest element is " << kthSmallest(arr, n, k);
    return 0;
}

```

Output:

K'th smallest element is 5

Method 4 (QuickSelect)

This is an optimization over method 1 if **QuickSort** is used as a sorting algorithm in first step. In QuickSort, we pick a pivot element, then move the pivot element to its correct position and partition the array around it. The idea is, not to do complete quicksort, but stop at the point where pivot itself is k'th smallest element. Also, not to recur for both left and right sides of pivot, but recur for one of them according to the position of pivot. The worst case time complexity of this method is $O(n^2)$, but it works in $O(n)$ on average.

```

#include<iostream>
#include<climits>
using namespace std;

int partition(int arr[], int l, int r);

// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method. ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)

```

```

{
    // Partition the array around last element and get
    // position of pivot element in sorted array
    int pos = partition(arr, l, r);

    // If position is same as k
    if (pos-1 == k-1)
        return arr[pos];
    if (pos-1 > k-1) // If position is more, recur for left subarray
        return kthSmallest(arr, l, pos-1, k);

    // Else recur for right subarray
    return kthSmallest(arr, pos+1, r, k-pos+l-1);
}

// If k is more than number of elements in array
return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort(). It considers the last
// element as pivot and moves all smaller element to left of it
// and greater elements to right
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k);
    return 0;
}

```

Output:

K'th smallest element is 5

There are two more solutions which are better than above discussed ones: One solution is to do randomized version of quickSelect() and other solution is worst case linear time algorithm (see the following posts).

[K'th Smallest/Largest Element in Unsorted Array | Set 2 \(Expected Linear Time\)](#)

[K'th Smallest/Largest Element in Unsorted Array | Set 3 \(Worst Case Linear Time\)](#)

References:

<http://www.ics.uci.edu/~eppstein/161/960125.html>

http://www.cs.rit.edu/~ib/Classes/CS515_Spring12-13/Slides/022-SelectMasterThm.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Heap](#) [Searching](#) [Order-Statistics](#)

Related Posts:

- Sum of all elements between k1'th and k2'th smallest elements
- Minimum sum of two numbers formed from digits of an array
- Heap using STL C++
- Convert min Heap to max Heap
- K-ary Heap
- Check if a given Binary Tree is Heap
- Fibonacci Heap | Set 1 (Introduction)
- How to check if a given array represents a Binary Heap?

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 50 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)

We recommend to read following post as a prerequisite of this post.

[K'th Smallest/Largest Element in Unsorted Array | Set 1](#)

Given an array and a number k where k is smaller than size of array, we need to find the k'th smallest element in the given array. It is given that all array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
```

Output: 7

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
```

Output: 10

We have discussed three different solutions [here](#).

In this post method 4 is discussed which is mainly an extension of method 3 (QuickSelect) discussed in the [previous post](#). The idea is to randomly pick a pivot element. To implement randomized partition, we use a random function, `rand()` to generate index between l and r, swap the element at randomly generated index with the last element, and finally call the standard partition process which uses last element as pivot.

Following is implementation of above Randomized QuickSelect.

```
// C++ implementation of randomized quickSelect
#include<iostream>
#include<climits>
#include<cstdlib>
using namespace std;

int randomPartition(int arr[], int l, int r);

// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method. ASSUMPTION: ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        // Partition the array around last element and get its index
        int pi = randomPartition(arr, l, r);

        // If k is same as partition index in arr[l..pi]
        if (k == pi)
            return arr[pi];
        // If k is smaller than partition index
        if (k < pi)
            return kthSmallest(arr, l, pi - 1, k);
        // Else k must be greater than partition index
        else
            return kthSmallest(arr, pi + 1, r, k - pi + l);
    }
}
```

```

// Partition the array around a random element and
// get position of pivot element in sorted array
int pos = randomPartition(arr, l, r);

// If position is same as k
if (pos-1 == k-1)
    return arr[pos];
if (pos-1 > k-1) // If position is more, recur for left subarray
    return kthSmallest(arr, l, pos-1, k);

// Else recur for right subarray
return kthSmallest(arr, pos+1, r, k-pos+1-1);
}

// If k is more than number of elements in array
return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort(). It considers the last
// element as pivot and moves all smaller element to left of it and
// greater elements to right. This function is used by randomPartition()
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Picks a random pivot element between l and r and partitions
// arr[l..r] around the randomly picked element using partition()
int randomPartition(int arr[], int l, int r)
{
    int n = r-l+1;
    int pivot = rand() % n;
    swap(&arr[l + pivot], &arr[r]);
    return partition(arr, l, r);
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to find k'th smallest element in expected
// linear time
class KthSmallest
{
    // This function returns k'th smallest element in arr[l..r]

```

```

// using QuickSort based method. ASSUMPTION: ALL ELEMENTS
// IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        // Partition the array around a random element and
        // get position of pivot element in sorted array
        int pos = randomPartition(arr, l, r);

        // If position is same as k
        if (pos-1 == k-1)
            return arr[pos];

        // If position is more, recur for left subarray
        if (pos-1 > k-1)
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }

    // If k is more than number of elements in array
    return Integer.MAX_VALUE;
}

// Utility method to swap arr[i] and arr[j]
void swap(int arr[], int i, int j)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

// Standard partition process of QuickSort(). It considers
// the last element as pivot and moves all smaller element
// to left of it and greater elements to right. This function
// is used by randomPartition()
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(arr, i, j);
            i++;
        }
    }
    swap(arr, i, r);
    return i;
}

// Picks a random pivot element between l and r and
// partitions arr[l..r] around the randomly picked
// element using partition()
int randomPartition(int arr[], int l, int r)
{
    int n = r-l+1;
    int pivot = (int)(Math.random()) % n;
    swap(arr, l + pivot, r);
    return partition(arr, l, r);
}

// Driver method to test above
public static void main(String args[])
{
    KthSmallest ob = new KthSmallest();
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = arr.length, k = 3;
    System.out.println("K'th smallest element is " +
                       ob.kthSmallest(arr, 0, n-1, k));
}

```

```
}
```

*/*This code is contributed by Rajat Mishra*/*

[Run on IDE](#)

Output:

```
K'th smallest element is 5
```

Time Complexity:

The worst case time complexity of the above solution is still $O(n^2)$. In worst case, the randomized function may always pick a corner element. The expected time complexity of above randomized QuickSelect is $\Theta(n)$, see [CLRS book](#) or [MIT video lecture](#) for proof. The assumption in the analysis is, random number generator is equally likely to generate any number in the input range.

Sources:

[MIT Video Lecture on Order Statistics, Median](#)

[Introduction to Algorithms](#) by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

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



Download

Free Download

[unzipper.com](#)



GATE CS Corner Company Wise Coding Practice

[Randomized](#) [Searching](#) [Order-Statistics](#) [Quick Sort](#)

Related Posts:

- Program to generate CAPTCHA and verify user
- Find an index of maximum occurring element with equal probability
- Implement rand12() using rand6() in one line
- Implement rand3() using rand2()
- Generate 0 and 1 with 25% and 75% probability

- Randomized Algorithms | Set 0 (Mathematical Background)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)
- Primality Test | Set 2 (Fermat Method)

(Login to Rate and Mark)

3.6

Average Difficulty : **3.6/5.0**
Based on **29** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

We recommend to read following posts as a prerequisite of this post.

[K'th Smallest/Largest Element in Unsorted Array | Set 1](#)

[K'th Smallest/Largest Element in Unsorted Array | Set 2 \(Expected Linear Time\)](#)

Given an array and a number k where k is smaller than size of array, we need to find the k'th smallest element in the given array. It is given that all array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7
```

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

In [previous post](#), we discussed an expected linear time algorithm. In this post, a worst case linear time method is discussed. *The idea in this new method is similar to quickSelect(), we get worst case linear time by selecting a pivot that divides array in a balanced way (there are not very few elements on one side and many on other side).* After the array is divided in a balanced way, we apply the same steps as used in quickSelect() to decide whether to go left or right of pivot.

Following is complete algorithm.

```
kthSmallest(arr[0..n-1], k)
1) Divide arr[] into [n/5] groups where size of each group is 5
   except possibly the last group which may have less than 5 elements.

2) Sort the above created [n/5] groups and find median
   of all groups. Create an auxiliary array 'median[]' and store medians
   of all [n/5] groups in this median array.

// Recursively call this method to find median of median[0..[n/5]-1]
3) medOfMed = kthSmallest(median[0..[n/5]-1], [n/10])

4) Partition arr[] around medOfMed and obtain its position.
   pos = partition(arr, n, medOfMed)
```

```

5) If pos == k return medOfMed
6) If pos < k return kthSmallest(arr[1..pos-1], k)
7) If pos > k return kthSmallest(arr[pos+1..r], k-pos+1-1)

```

In above algorithm, last 3 steps are same as algorithm in [previous post](#). The first four steps are used to obtain a good point for partitioning the array (to make sure that there are not too many elements either side of pivot).

Following is C++ implementation of above algorithm.

```

// C++ implementation of worst case linear time algorithm
// to find k'th smallest element
#include<iostream>
#include<algorithm>
#include<climits>
using namespace std;

int partition(int arr[], int l, int r, int k);

// A simple function to find median of arr[]. This is called
// only for an array of size 5 in this program.
int findMedian(int arr[], int n)
{
    sort(arr, arr+n); // Sort the array
    return arr[n/2]; // Return middle element
}

// Returns k'th smallest element in arr[l..r] in worst case
// linear time. ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        int n = r-l+1; // Number of elements in arr[l..r]

        // Divide arr[] in groups of size 5, calculate median
        // of every group and store it in median[] array.
        int i, median[(n+4)/5]; // There will be floor((n+4)/5) groups;
        for (i=0; i<n/5; i++)
            median[i] = findMedian(arr+l+i*5, 5);
        if (i*5 < n) //For last group with less than 5 elements
        {
            median[i] = findMedian(arr+l+i*5, n%5);
            i++;
        }

        // Find median of all medians using recursive call.
        // If median[] has only one element, then no need
        // of recursive call
        int medOfMed = (i == 1)? median[i-1]:
                           kthSmallest(median, 0, i-1, i/2);

        // Partition the array around a random element and
        // get position of pivot element in sorted array
        int pos = partition(arr, l, r, medOfMed);

        // If position is same as k
        if (pos-1 == k-1)
            return arr[pos];
        if (pos-1 > k-1) // If position is more, recur for left
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+1-1);
    }

    // If k is more than number of elements in array
    return INT_MAX;
}

void swap(int *a, int *b)
{
}

```

```

int temp = *a;
*a = *b;
*b = temp;
}

// It searches for x in arr[l..r], and partitions the array
// around x.
int partition(int arr[], int l, int r, int x)
{
    // Search for x in arr[l..r] and move it to end
    int i;
    for (i=1; i<r; i++)
        if (arr[i] == x)
            break;
    swap(&arr[i], &arr[r]);

    // Standard partition algorithm
    i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is "
         << kthSmallest(arr, 0, n-1, k);
    return 0;
}

```

Run on IDE

Output:

K'th smallest element is 5

Time Complexity:

The worst case time complexity of the above algorithm is O(n). Let us analyze all steps.

The steps 1) and 2) take O(n) time as finding median of an array of size 5 takes O(1) time and there are n/5 arrays of size 5.

The step 3) takes T(n/5) time. The step 4) is standard partition and takes O(n) time.

The interesting steps are 6) and 7). At most, one of them is executed. These are recursive steps. What is the worst case size of these recursive calls. The answer is maximum number of elements greater than medOfMed (obtained in step 3) or maximum number of elements smaller than medOfMed.

How many elements are greater than medOfMed and how many are smaller?

At least half of the medians found in step 2 are greater than or equal to medOfMed. Thus, at least half of the n/5 groups contribute 3 elements that are greater than medOfMed, except for the one group that has fewer than 5 elements. Therefore, the number of elements greater than medOfMed is at least.

$$3 \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$$

Similarly, the number of elements that are less than medOfMed is at least $3n/10 - 6$. In the worst case, the function recurs for at most $n - (3n/10 - 6)$ which is $7n/10 + 6$ elements.

Note that $7n/10 + 6 < n$ for $n > 20$ and that any input of 80 or fewer elements requires $O(1)$ time. We can therefore obtain the recurrence

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 80, \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n > 80. \end{cases}$$

We show that the running time is linear by substitution. Assume that $T(n) \leq cn$ for some constant c and all $n > 80$. Substituting this inductive hypothesis into the right-hand side of the recurrence yields

$$\begin{aligned} T(n) &\leq cn/5 + c(7n/10 + 6) + O(n) \\ &\leq cn/5 + c + 7cn/10 + 6c + O(n) \\ &\leq 9cn/10 + 7c + O(n) \\ &\leq cn, \end{aligned}$$

since we can pick c large enough so that $c(n/10 - 7)$ is larger than the function described by the $O(n)$ term for all $n > 80$. The worst-case running time of is therefore linear (Source: <http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap10.htm>).

Note that the above algorithm is linear in worst case, but the constants are very high for this algorithm. Therefore, this algorithm doesn't work well in practical situations, [randomized quickSelect](#) works much better and preferred.

Sources:

[MIT Video Lecture on Order Statistics, Median](#)

[Introduction to Algorithms](#) by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

<http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap10.htm>

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



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Searching

Order-Statistics

Related Posts:

- Second minimum element using minimum comparisons
- Interpolation Search
- Jump Search
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

(Login to Rate and Mark)

4.3

Average Difficulty : **4.3/5.0**
Based on **45** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find the closest pair from two sorted arrays

Given two sorted arrays and a number x, find the pair whose sum is closest to x and **the pair has an element from each array.**

We are given two arrays ar1[0...m-1] and ar2[0..n-1] and a number x, we need to find the pair ar1[i] + ar2[j] such that absolute value of (ar1[i] + ar2[j] – x) is minimum.

Example:

```
Input: ar1[] = {1, 4, 5, 7};
       ar2[] = {10, 20, 30, 40};
       x = 32
Output: 1 and 30
```

```
Input: ar1[] = {1, 4, 5, 7};
       ar2[] = {10, 20, 30, 40};
       x = 50
Output: 7 and 40
```

We strongly recommend to minimize your browser and try this yourself first.

A **Simple Solution** is to run two loops. The outer loop considers every element of first array and inner loop checks for the pair in second array. We keep track of minimum difference between ar1[i] + ar2[j] and x.

We can do it **in O(n) time** using following steps.

1) Merge given two arrays into an auxiliary array of size m+n using **merge process of merge sort**. While merging keep another boolean array of size m+n to indicate whether the current element in merged array is from ar1[] or ar2[].

2) Consider the merged array and use the **linear time algorithm to find the pair with sum closest to x**. One extra thing we need to consider only those pairs which have one element from ar1[] and other from ar2[], we use the boolean array for this purpose.

Can we do it in a single pass and O(1) extra space?

The idea is to start from left side of one array and right side of another array, and use the algorithm same as step 2 of above approach. Following is detailed algorithm.

- 1) Initialize a variable diff as infinite (Diff is used to store the difference between pair and x). We need to find the minimum diff.
- 2) Initialize two index variables l and r in the given sorted array.
 - (a) Initialize first to the leftmost index in ar1: l = 0
 - (b) Initialize second the rightmost index in ar2: r = n-1

- 3) Loop while $l < m$ and $r \geq 0$
 - (a) If $\text{abs}(\text{ar1}[l] + \text{ar2}[r] - \text{sum}) < \text{diff}$ then
update diff and result
 - (b) Else if($\text{ar1}[l] + \text{ar2}[r] < \text{sum}$) then $l++$
 - (c) Else $r--$
- 4) Print the result.

Following is C++ implementation of this approach.

```
// C++ program to find the pair from two sorted arrays such
// that the sum of pair is closest to a given number x
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

// ar1[0..m-1] and ar2[0..n-1] are two given sorted arrays
// and x is given number. This function prints the pair from
// both arrays such that the sum of the pair is closest to x.
void printClosest(int ar1[], int ar2[], int m, int n, int x)
{
    // Initialize the diff between pair sum and x.
    int diff = INT_MAX;

    // res_l and res_r are result indexes from ar1[] and ar2[]
    // respectively
    int res_l, res_r;

    // Start from left side of ar1[] and right side of ar2[]
    int l = 0, r = n-1;
    while (l < m && r >= 0)
    {
        // If this pair is closer to x than the previously
        // found closest, then update res_l, res_r and diff
        if (abs(ar1[l] + ar2[r] - x) < diff)
        {
            res_l = l;
            res_r = r;
            diff = abs(ar1[l] + ar2[r] - x);
        }

        // If sum of this pair is more than x, move to smaller
        // side
        if (ar1[l] + ar2[r] > x)
            r--;
        else // move to the greater side
            l++;
    }

    // Print the result
    cout << "The closest pair is [" << ar1[res_l] << ", "
        << ar2[res_r] << "] \n";
}

// Driver program to test above functions
int main()
{
    int ar1[] = {1, 4, 5, 7};
    int ar2[] = {10, 20, 30, 40};
    int m = sizeof(ar1)/sizeof(ar1[0]);
    int n = sizeof(ar2)/sizeof(ar2[0]));
    int x = 38;
    printClosest(ar1, ar2, m, n, x);
    return 0;
}
```

Run on IDE

Java

```
// Java program to find closest pair in an array
class ClosestPair
{
    // ar1[0..m-1] and ar2[0..n-1] are two given sorted
    // arrays/ and x is given number. This function prints
    // the pair from both arrays such that the sum of the
    // pair is closest to x.
    void printClosest(int ar1[], int ar2[], int m, int n, int x)
    {
        // Initialize the diff between pair sum and x.
        int diff = Integer.MAX_VALUE;

        // res_l and res_r are result indexes from ar1[] and ar2[]
        // respectively
        int res_l = 0, res_r = 0;

        // Start from left side of ar1[] and right side of ar2[]
        int l = 0, r = n-1;
        while (l < m && r >= 0)
        {
            // If this pair is closer to x than the previously
            // found closest, then update res_l, res_r and diff
            if (Math.abs(ar1[l] + ar2[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.abs(ar1[l] + ar2[r] - x);
            }

            // If sum of this pair is more than x, move to smaller
            // side
            if (ar1[l] + ar2[r] > x)
                r--;
            else // move to the greater side
                l++;
        }

        // Print the result
        System.out.print("The closest pair is [" + ar1[res_l] +
                         ", " + ar2[res_r] + "]");
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        ClosestPair ob = new ClosestPair();
        int ar1[] = {1, 4, 5, 7};
        int ar2[] = {10, 20, 30, 40};
        int m = ar1.length;
        int n = ar2.length;
        int x = 38;
        ob.printClosest(ar1, ar2, m, n, x);
    }
}
/*This code is contributed by Rajat Mishra */
```

Run on IDE

Output:

The closest pair is [7, 30]

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



GATE CS Corner Company Wise Coding Practice

Searching

Related Posts:

- Second minimum element using minimum comparisons
- [Interpolation Search](#)
- [Jump Search](#)
- Third largest element in an array of distinct elements
- Floor in a Sorted Array
- Fibonacci Search
- Find the element before which all the elements are smaller than it, and after which all are greater
- Find three closest elements from given three sorted arrays

(Login to Rate and Mark)

3.1

Average Difficulty : **3.1/5.0**
Based on **39** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find common elements in three sorted arrays

Given three arrays sorted in non-decreasing order, print all common elements in these arrays.

Examples:

```
ar1[] = {1, 5, 10, 20, 40, 80}
ar2[] = {6, 7, 20, 80, 100}
ar3[] = {3, 4, 15, 20, 30, 70, 80, 120}
Output: 20, 80
```

```
ar1[] = {1, 5, 5}
ar2[] = {3, 4, 5, 5, 10}
ar3[] = {5, 5, 10, 20}
Output: 5, 5
```

A simple solution is to first find [intersection of two arrays](#) and store the intersection in a temporary array, then find the intersection of third array and temporary array. Time complexity of this solution is $O(n_1 + n_2 + n_3)$ where n_1 , n_2 and n_3 are sizes of $ar1[]$, $ar2[]$ and $ar3[]$ respectively.

The above solution requires extra space and two loops, we can find the common elements using a single loop and without extra space. The idea is similar to [intersection of two arrays](#). Like two arrays loop, we run a loop and traverse three arrays.

Let the current element traversed in $ar1[]$ be x , in $ar2[]$ be y and in $ar3[]$ be z . We can have following cases inside the loop.

- 1) If x , y and z are same, we can simply print any of them as common element and move ahead in all three arrays.
- 2) Else If $x < y$, we can move ahead in $ar1[]$ as x cannot be a common element
- 3) Else If $y < z$, we can move ahead in $ar2[]$ as y cannot be a common element
- 4) Else (We reach here when $x > y$ and $y > z$), we can simply move ahead in $ar3[]$ as z cannot be a common element.

Following are implementations of the above idea.

```
// C++ program to print common elements in three arrays
#include <iostream>
using namespace std;

// This function prints common elements in ar1
void findCommon(int ar1[], int ar2[], int ar3[], int n1, int n2, int n3)
{
    // Initialize starting indexes for ar1[], ar2[] and ar3[]
    int i = 0, j = 0, k = 0;
```

```

// Iterate through three arrays while all arrays have elements
while (i < n1 && j < n2 && k < n3)
{
    // If x = y and y = z, print any of them and move ahead
    // in all arrays
    if (ar1[i] == ar2[j] && ar2[j] == ar3[k])
    {   cout << ar1[i] << " ";   i++; j++; k++; }

    // x < y
    else if (ar1[i] < ar2[j])
        i++;

    // y < z
    else if (ar2[j] < ar3[k])
        j++;

    // We reach here when x > y and z < y, i.e., z is smallest
    else
        k++;
}

// Driver program to test above function
int main()
{
    int ar1[] = {1, 5, 10, 20, 40, 80};
    int ar2[] = {6, 7, 20, 80, 100};
    int ar3[] = {3, 4, 15, 20, 30, 70, 80, 120};
    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    int n3 = sizeof(ar3)/sizeof(ar3[0]);

    cout << "Common Elements are ";
    findCommon(ar1, ar2, ar3, n1, n2, n3);
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python function to print common elements in three sorted arrays
def findCommon(ar1, ar2, ar3, n1, n2, n3):

    # Initialize starting indexes for ar1[], ar2[] and ar3[]
    i, j, k = 0, 0, 0

    # Iterate through three arrays while all arrays have elements
    while (i < n1 and j < n2 and k < n3):

        # If x = y and y = z, print any of them and move ahead
        # in all arrays
        if (ar1[i] == ar2[j] and ar2[j] == ar3[k]):
            print ar1[i],
            i += 1
            j += 1
            k += 1

        # x < y
        elif ar1[i] < ar2[j]:
            i += 1

        # y < z
        elif ar2[j] < ar3[k]:
            j += 1

        # We reach here when x > y and z < y, i.e., z is smallest
        else:
            k += 1

# Driver program to check above function

```

```

ar1 = [1, 5, 10, 20, 40, 80]
ar2 = [6, 7, 20, 80, 100]
ar3 = [3, 4, 15, 20, 30, 70, 80, 120]
n1 = len(ar1)
n2 = len(ar2)
n3 = len(ar3)
print "Common elements are",
findCommon(ar1, ar2, ar3, n1, n2, n3)

# This code is contributed by _Devesh Agrawal_

```

[Run on IDE](#)

Java

```

// Java program to find common elements in three arrays
class FindCommon
{
    // This function prints common elements in ar1
    void findCommon(int ar1[], int ar2[], int ar3[])
    {
        // Initialize starting indexes for ar1[], ar2[] and ar3[]
        int i = 0, j = 0, k = 0;

        // Iterate through three arrays while all arrays have elements
        while (i < ar1.length && j < ar2.length && k < ar3.length)
        {
            // If x = y and y = z, print any of them and move ahead
            // in all arrays
            if (ar1[i] == ar2[j] && ar2[j] == ar3[k])
            {
                System.out.print(ar1[i]+ " ");
                i++; j++; k++;
            }

            // x < y
            else if (ar1[i] < ar2[j])
                i++;

            // y < z
            else if (ar2[j] < ar3[k])
                j++;

            // We reach here when x > y and z < y, i.e., z is smallest
            else
                k++;
        }
    }

    // Driver code to test above
    public static void main(String args[])
    {
        FindCommon ob = new FindCommon();

        int ar1[] = {1, 5, 10, 20, 40, 80};
        int ar2[] = {6, 7, 20, 80, 100};
        int ar3[] = {3, 4, 15, 20, 30, 70, 80, 120};

        System.out.print("Common elements are ");
        ob.findCommon(ar1, ar2, ar3);
    }
}
/*This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

Common Elements are 20 80

Time complexity of the above solution is $O(n_1 + n_2 + n_3)$. In worst case, the largest sized array may have all small elements and middle sized array has all middle elements.

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



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Arrays

Related Posts:

- Maximum sum subarray removing at most one element
- Steps to return to $\{1, 2, \dots, n\}$ with specified movements
- Position of an element after stable sort
- Move all negative elements to end in order with extra space allowed
- Minimize the sum of product of two arrays with permutations allowed
- Subarrays with distinct elements
- Sort an array according to absolute difference with given value
- Sum of all elements between k_1 'th and k_2 'th smallest elements

(Login to Rate and Mark)

1.8

Average Difficulty : 1.8/5.0
Based on 65 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Given a sorted array and a number x, find the pair in array whose sum is closest to x

Given a sorted array and a number x, find a pair in array whose sum is closest to x.

Examples:

```
Input: arr[] = {10, 22, 28, 29, 30, 40}, x = 54
Output: 22 and 30
```

```
Input: arr[] = {1, 3, 4, 7, 10}, x = 15
Output: 4 and 10
```

A simple solution is to consider every pair and keep track of closest pair (absolute difference between pair sum and x is minimum). Finally print the closest pair. Time complexity of this solution is $O(n^2)$

An efficient solution can find the pair in $O(n)$ time. The idea is similar to method 2 of [this](#) post. Following is detailed algorithm.

- 1) Initialize a variable diff as infinite (Diff is used to store the difference between pair and x). We need to find the minimum diff.
- 2) Initialize two index variables l and r in the given sorted array.
 - (a) Initialize first to the leftmost index: $l = 0$
 - (b) Initialize second the rightmost index: $r = n-1$
- 3) Loop while $l < r$.
 - (a) If $\text{abs}(\text{arr}[l] + \text{arr}[r] - \text{sum}) < \text{diff}$ then update diff and result
 - (b) Else if($\text{arr}[l] + \text{arr}[r] < \text{sum}$) then $l++$
 - (c) Else $r--$

Following is C++ implementation of above algorithm.

```
// Simple C++ program to find the pair with sum closest to a given no.
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

// Prints the pair with sum closest to x
void printClosest(int arr[], int n, int x)
{
    int res_l, res_r; // To store indexes of result pair

    // Initialize left and right indexes and difference between
    // pair sum and x
    int l = 0, r = n-1, diff = INT_MAX;

    // While there are elements between l and r
    while (r > l)
```

```

    // Check if this pair is closer than the closest pair so far
    if (abs(arr[l] + arr[r] - x) < diff)
    {
        res_l = l;
        res_r = r;
        diff = abs(arr[l] + arr[r] - x);
    }

    // If this pair has more sum, move to smaller values.
    if (arr[l] + arr[r] > x)
        r--;
    else // Move to larger values
        l++;
}

cout <<" The closest pair is " << arr[res_l] << " and " << arr[res_r];
}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 22, 28, 29, 30, 40}, x = 54;
    int n = sizeof(arr)/sizeof(arr[0]);
    printClosest(arr, n, x);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to find pair with sum closest to x
import java.io.*;
import java.util.*;
import java.lang.Math;

class CloseSum {

    // Prints the pair with sum closest to x
    static void printClosest(int arr[], int n, int x)
    {
        int res_l=0, res_r=0; // To store indexes of result pair

        // Initialize left and right indexes and difference between
        // pair sum and x
        int l = 0, r = n-1, diff = Integer.MAX_VALUE;

        // While there are elements between l and r
        while (r > l)
        {
            // Check if this pair is closer than the closest pair so far
            if (Math.abs(arr[l] + arr[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.abs(arr[l] + arr[r] - x);
            }

            // If this pair has more sum, move to smaller values.
            if (arr[l] + arr[r] > x)
                r--;
            else // Move to larger values
                l++;
        }

        System.out.println(" The closest pair is "+arr[res_l]+" and "+ arr[res_r]);
    }
}

// Driver program to test above function

```

```

public static void main(String[] args)
{
    int arr[] = {10, 22, 28, 29, 30, 40}, x = 54;
    int n = arr.length;
    printClosest(arr, n, x);
}
/*This code is contributed by Devesh Agrawal*/

```

[Run on IDE](#)**Output:**

```
The closest pair is 22 and 30
```

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

GATE CS Corner



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

(Login to Rate and Mark)

Average Rating : **3/5.0**

Average Difficulty : **2.4/5.0**

2.4

Based on **34** vote(s)



Add to TODO List



Mark as DONE

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Count 1's in a sorted binary array

Given a binary array sorted in non-increasing order, count the number of 1's in it.

Examples:

```
Input: arr[] = {1, 1, 0, 0, 0, 0, 0}
Output: 2
```

```
Input: arr[] = {1, 1, 1, 1, 1, 1, 1}
Output: 7
```

```
Input: arr[] = {0, 0, 0, 0, 0, 0, 0}
Output: 0
```

A simple solution is to linearly traverse the array. The time complexity of the simple solution is $O(n)$. We can use [Binary Search](#) to find count in $O(\log n)$ time. The idea is to look for last occurrence of 1 using Binary Search. Once we find the index last occurrence, we return index + 1 as count.

The following is C++ implementation of above idea.

```
// C++ program to count one's in a boolean array
#include <iostream>
using namespace std;

/* Returns counts of 1's in arr[low..high]. The array is
   assumed to be sorted in non-increasing order */
int countOnes(bool arr[], int low, int high)
{
    if (high >= low)
    {
        // get the middle index
        int mid = low + (high - low)/2;

        // check if the element at middle index is last 1
        if ((mid == high || arr[mid+1] == 0) && (arr[mid] == 1))
            return mid+1;

        // If element is not last 1, recur for right side
        if (arr[mid] == 1)
            return countOnes(arr, (mid + 1), high);

        // else recur for left side
        return countOnes(arr, low, (mid -1));
    }
    return 0;
}

/* Driver program to test above functions */
int main()
{
    bool arr[] = {1, 1, 1, 1, 0, 0, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```

cout << "Count of 1's in given array is " << countOnes(arr, 0, n-1);
return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to count one's in a boolean array

# Returns counts of 1's in arr[low..high]. The array is
# assumed to be sorted in non-increasing order
def countOnes(arr,low,high):

    if high>=low:

        # get the middle index
        mid = low + (high-low)/2

        # check if the element at middle index is last 1
        if ((mid == high or arr[mid+1]==0) and (arr[mid]==1)):
            return mid+1

        # If element is not last 1, recur for right side
        if arr[mid]==1:
            return countOnes(arr, (mid+1), high)

        # else recur for left side
        return countOnes(arr, low, mid-1)

    return 0

# Driver function
arr=[1, 1, 1, 1, 0, 0]
print "Count of 1's in given array is",countOnes(arr, 0 , len(arr)-1)

# This code is contributed by _Devesh Agrawal_

```

[Run on IDE](#)

Java

```

// Java program to count 1's in a sorted array
class CountOnes
{
    /* Returns counts of 1's in arr[low..high]. The
       array is assumed to be sorted in non-increasing
       order */
    int countOnes(int arr[], int low, int high)
    {
        if (high >= low)
        {
            // get the middle index
            int mid = low + (high - low)/2;

            // check if the element at middle index is last 1
            if ( (mid == high || arr[mid+1] == 0) &&
                (arr[mid] == 1))
                return mid+1;

            // If element is not last 1, recur for right side
            if (arr[mid] == 1)
                return countOnes(arr, (mid + 1), high);

            // else recur for left side
            return countOnes(arr, low, (mid -1));
        }
        return 0;
    }
}

```

```
}

/* Driver program to test above functions */
public static void main(String args[])
{
    CountOnes ob = new CountOnes();
    int arr[] = {1, 1, 1, 1, 0, 0, 0};
    int n = arr.length;
    System.out.println("Count of 1's in given array is " +
        ob.countOnes(arr, 0, n-1) );
}

/* This code is contributed by Rajat Mishra */
```

[Run on IDE](#)

Output:

```
Count of 1's in given array is 4
```

Time complexity of the above solution is O(Logn)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

(Login to Rate and Mark)

Average Rating : **3.5/5.0**

Average Difficulty : **1.9/5.0**

1.9

Based on **26** vote(s)



Add to TODO List



Mark as DONE

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Binary Insertion Sort

We can use binary search to reduce the number of comparisons in [normal insertion sort](#). Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration.

In normal insertion, sort it takes $O(i)$ (at i th iteration) in worst case. we can reduce it to $O(\log i)$ by using [binary search](#).

```
// C program for implementation of binary insertion sort
#include <stdio.h>

// A binary search based function to find the position
// where item should be inserted in a[low..high]
int binarySearch(int a[], int item, int low, int high)
{
    if (high <= low)
        return (item > a[low])? (low + 1): low;

    int mid = (low + high)/2;

    if(item == a[mid])
        return mid+1;

    if(item > a[mid])
        return binarySearch(a, item, mid+1, high);
    return binarySearch(a, item, low, mid-1);
}

// Function to sort an array a[] of size 'n'
void insertionSort(int a[], int n)
{
    int i, loc, j, k, selected;

    for (i = 1; i < n; ++i)
    {
        j = i - 1;
        selected = a[i];

        // find location where selected sould be insereted
        loc = binarySearch(a, selected, 0, j);

        // Move all elements after location to create space
        while (j >= loc)
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = selected;
    }
}

// Driver program to test above function
int main()
{
    int a[] = {37, 23, 0, 17, 12, 72, 31,
              46, 100, 88, 54};
    int n = sizeof(a)/sizeof(a[0]), i;

    insertionSort(a, n);
}
```

```

printf("Sorted array: \n");
for (i = 0; i < n; i++)
    printf("%d ",a[i]);

return 0;
}

```

[Run on IDE](#)

Output:

```

Sorted array:
0 12 17 23 31 37 46 54 72 88 100

```

Time Complexity: The algorithm as a whole still has a running worst case running time of $O(n^2)$ because of the series of swaps required for each insertion.

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

GATE CS Corner

The advertisement features a dark red background. At the top right is a small icon with a magnifying glass and a 'X'. Below it, the text "Introduction to Computer Science" is displayed in large white font. Underneath that, "ENROLLMENT IS OPEN TO EVERYONE" is written in smaller white font. In the center is a white button with the text "Start Now" in red. At the bottom left is the edX logo with the website "www.edx.org". To its right is the Harvard University logo with the text "HARVARD UNIVERSITY".

See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms Searching and Sorting

(Login to Rate and Mark)

Average Rating : **2.6/5.0**

Based on 3 vote(s)



Average Difficulty : **2.5/5.0**

2.5

Based on 13 vote(s)



Add to TODO List

Mark as DONE

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Insertion Sort for Singly Linked List

We have discussed [Insertion Sort for arrays](#). In this article same for linked list is discussed.

Below is simple insertion sort algorithm for linked list.

- 1) Create an empty sorted (or result) list
- 2) Traverse the given list, do following for every node.
 -a) Insert current node in sorted way in sorted or result list.
- 3) Change head of given linked list to head of sorted (or result) list.

The main step is (2.a) which has been covered in below post.

[Sorted Insert for Singly Linked List](#)

Below is C implementation of above algorithm

```
/* C program for insertion sort on a linked list */
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

// Function to insert a given node in a sorted linked list
void sortedInsert(struct node**, struct node*);

// function to sort a singly linked list using insertion sort
void insertionSort(struct node **head_ref)
{
    // Initialize sorted linked list
    struct node *sorted = NULL;

    // Traverse the given linked list and insert every
    // node to sorted
    struct node *current = *head_ref;
    while (current != NULL)
    {
        // Store next for next iteration
        struct node *next = current->next;

        // insert current in sorted linked list
        sortedInsert(&sorted, current);

        // Update current
        current = next;
    }

    // Update head_ref to point to sorted linked list
    *head_ref = sorted;
}

/* function to insert a new_node in a list. Note that this
   function expects a pointer to head_ref as this can modify the
```

```

head of the input linked list (similar to push())*/
void sortedInsert(struct node** head_ref, struct node* new_node)
{
    struct node* current;
    /* Special case for the head end */
    if (*head_ref == NULL || (*head_ref)->data >= new_node->data)
    {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
    else
    {
        /* Locate the node before the point of insertion */
        current = *head_ref;
        while (current->next!=NULL &&
               current->next->data < new_node->data)
        {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}

/* BELOW FUNCTIONS ARE JUST UTILITY TO TEST sortedInsert */

/* Function to print linked list */
void printList(struct node *head)
{
    struct node *temp = head;
    while(temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

/* A utility function to insert a node at the beginning of linked list */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node = new 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;
}

// Driver program to test above functions
int main()
{
    struct node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    printf("Linked List before sorting \n");
    printList(a);

    insertionSort(&a);

    printf("\nLinked List after sorting \n");
    printList(a);

    return 0;
}

```

```
Linked List before sorting
```

```
30 3 4 20 5
```

```
Linked List after sorting
```

```
3 4 5 20 30
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner



Free Download

Zip, Unzip or Open Any File.

Unzipper



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: [Linked List](#)

([Login](#) to Rate and Mark)

Average Rating : **4.6/5.0**

Average Difficulty : **3.3/5.0**

3.3

Based on **20** vote(s)

Based on **3** vote(s)



Add to TODO List



Mark as DONE

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

Practice

GATE CS

Placements

GeeksQuiz

Google™ Custom Search



Login/Register

Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists?

Why is Quick Sort preferred for arrays?

Below are recursive and iterative implementations of Quick Sort and Merge Sort for arrays.

[Recursive Quick Sort for array.](#)

[Iterative Quick Sort for arrays.](#)

[Recursive Merge Sort for arrays](#)

[Iterative Merge Sort for arrays](#)

Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires $O(N)$ extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have $O(N\log N)$ average complexity but the constants differ. For arrays, merge sort loses due to the use of extra $O(N)$ storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of $O(n\log n)$. The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Quick Sort is also a cache friendly sorting algorithm as it has good [locality of reference](#) when used for arrays.

Quick Sort is also [tail recursive](#), therefore tail call optimizations is done.

Why is Merge Sort preferred for Linked Lists?

Below are implementations of Quicksort and Mergesort for singly and doubly linked lists.

[Quick Sort for Doubly Linked List](#)

[Quick Sort for Singly Linked List](#)

[Merge Sort for Singly Linked List](#)

[Merge Sort for Doubly Linked List](#)

In case of [linked lists](#) the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in [linked list](#), we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of $A[0]$ be x then to access $A[i]$, we can directly access the memory at $(x + i * 4)$.

Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

Related Articles:

[Know Your Sorting Algorithm | Set 1 \(Sorting Weapons used by Programming Languages\)](#)

[Iterative Merge Sort](#)

[Iterative Quick Sort](#)

Thanks to [Sayan Mukhopadhyay](#) for providing initial draft for above article. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Sorting](#) [Merge Sort](#) [Quick Sort](#)

Related Posts:

- Position of an element after stable sort
- Minimize the sum of product of two arrays with permutations allowed
- Sort an array according to absolute difference with given value
- Number of sextuplets (or six values) that satisfy an equation
- Sum of all elements between k1'th and k2'th smallest elements
- Count minimum number of subsets (or subsequences) with consecutive numbers
- Find elements larger than half of the elements in an array
- Minimum swaps to make two arrays identical

(Login to Rate and Mark)

2.3

Average Difficulty : **2.3/5.0**
Based on **35** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

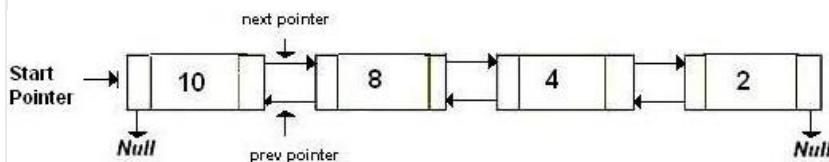


[Login/Register](#)

Merge Sort for Doubly Linked List

Given a doubly linked list, write a function to sort the doubly linked list in increasing order using merge sort.

For example, the following doubly linked list should be changed to 2<->4<->8<->10



We strongly recommend to minimize your browser and try this yourself first.

Merge sort for singly linked list is already discussed. The important change here is to modify the previous pointers also when merging two lists.

Below is the implementation of merge sort for doubly linked list.

```

// C program for merge sort on doubly linked list
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next, *prev;
};

struct node *split(struct node *head);

// Function to merge two linked lists
struct node *merge(struct node *first, struct node *second)
{
    // If first linked list is empty
    if (!first)
        return second;

    // If second linked list is empty
    if (!second)
        return first;

    // Pick the smaller value
    if (first->data < second->data)
    {
        first->next = merge(first->next, second);
        first->next->prev = first;
        first->prev = NULL;
        return first;
    }
}
  
```

```

else
{
    second->next = merge(first,second->next);
    second->next->prev = second;
    second->prev = NULL;
    return second;
}

// Function to do merge sort
struct node *mergeSort(struct node *head)
{
    if (!head || !head->next)
        return head;
    struct node *second = split(head);

    // Recur for left and right halves
    head = mergeSort(head);
    second = mergeSort(second);

    // Merge the two sorted halves
    return merge(head,second);
}

// A utility function to insert a new node at the
// beginning of doubly linked list
void insert(struct node **head, int data)
{
    struct node *temp =
        (struct node *)malloc(sizeof(struct node));
    temp->data = data;
    temp->next = temp->prev = NULL;
    if (!(*head))
        (*head) = temp;
    else
    {
        temp->next = *head;
        (*head)->prev = temp;
        (*head) = temp;
    }
}

// A utility function to print a doubly linked list in
// both forward and backward directions
void print(struct node *head)
{
    struct node *temp = head;
    printf("Forward Traversal using next pointer\n");
    while (head)
    {
        printf("%d ", head->data);
        temp = head;
        head = head->next;
    }
    printf("\nBackward Traversal using prev pointer\n");
    while (temp)
    {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
}

// Utility function to swap two integers
void swap(int *A, int *B)
{
    int temp = *A;
    *A = *B;
    *B = temp;
}

// Split a doubly linked list (DLL) into 2 DLLs of
// half sizes
struct node *split(struct node *head)
{
}

```

```

struct node *fast = head,*slow = head;
while (fast->next && fast->next->next)
{
    fast = fast->next->next;
    slow = slow->next;
}
struct node *temp = slow->next;
slow->next = NULL;
return temp;
}

// Driver program
int main(void)
{
    struct node *head = NULL;
    insert(&head,5);
    insert(&head,20);
    insert(&head,4);
    insert(&head,3);
    insert(&head,30);
    insert(&head,10);
    head = mergeSort(head);
    printf("\n\nLinked List after sorting\n");
    print(head);
    return 0;
}

```

Run on IDE

Java

```

// Java program to implement merge sort in singly linked list

// Linked List Class
class LinkedList {

    static Node head; // head of list

    /* Node Class */
    static class Node {

        int data;
        Node next, prev;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = prev = null;
        }
    }

    void print(Node node) {
        Node temp = node;
        System.out.println("Forward Traversal using next pointer");
        while (node != null) {
            System.out.print(node.data + " ");
            temp = node;
            node = node.next;
        }
        System.out.println("\nBackward Traversal using prev pointer");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.prev;
        }
    }

    // Split a doubly linked list (DLL) into 2 DLLs of
    // half sizes
    Node split(Node head) {
        Node fast = head, slow = head;
        while (fast.next != null && fast.next.next != null) {

```

```

        fast = fast.next.next;
        slow = slow.next;
    }
    Node temp = slow.next;
    slow.next = null;
    return temp;
}

Node mergeSort(Node node) {
    if (node == null || node.next == null) {
        return node;
    }
    Node second = split(node);

    // Recur for left and right halves
    node = mergeSort(node);
    second = mergeSort(second);

    // Merge the two sorted halves
    return merge(node, second);
}

// Function to merge two linked lists
Node merge(Node first, Node second) {
    // If first linked list is empty
    if (first == null) {
        return second;
    }

    // If second linked list is empty
    if (second == null) {
        return first;
    }

    // Pick the smaller value
    if (first.data < second.data) {
        first.next = merge(first.next, second);
        first.next.prev = first;
        first.prev = null;
        return first;
    } else {
        second.next = merge(first, second.next);
        second.next.prev = second;
        second.prev = null;
        return second;
    }
}

// Driver program to test above functions
public static void main(String[] args) {

    LinkedList list = new LinkedList();
    list.head = new Node(10);
    list.head.next = new Node(30);
    list.head.next.next = new Node(3);
    list.head.next.next.next = new Node(4);
    list.head.next.next.next.next = new Node(20);
    list.head.next.next.next.next.next = new Node(5);

    Node node = null;
    node = list.mergeSort(head);
    System.out.println("Linked list after sorting :");
    list.print(node);
}

// This code has been contributed by Mayank Jaiswal

```

[Run on IDE](#)

Python

```
# Program for merge sort on doubly linked list

# A node of the doubly linked list
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

class DoublyLinkedList:

    # Constructor for empty Doubly Linked List
    def __init__(self):
        self.head = None

    # Function to merge two linked list
    def merge(self, first, second):

        # If first linked list is empty
        if first is None:
            return second

        # If secon linked list is empty
        if second is None:
            return first

        # Pick the smaller value
        if first.data < second.data:
            first.next = self.merge(first.next, second)
            first.next.prev = first
            first.prev = None
            return first
        else:
            second.next = self.merge(first, second.next)
            second.next.prev = second
            second.prev = None
            return second

    # Function to do merge sort
    def mergeSort(self, tempHead):
        if tempHead is None:
            return tempHead
        if tempHead.next is None:
            return tempHead

        second = self.split(tempHead)

        # Recur for left and righ halves
        tempHead = self.mergeSort(tempHead)
        second = self.mergeSort(second)

        # Merge the two sorted halves
        return self.merge(tempHead, second)

    # Split the doubly linked list (DLL) into two DLLs
    # of half sizes
    def split(self, tempHead):
        fast = slow = tempHead
        while(True):
            if fast.next is None:
                break
            if fast.next.next is None:
                break
            fast = fast.next.next
            slow = slow.next

        temp = slow.next
        slow.next = None
```

```

return temp

# Given a reference to the head of a list and an
# integer, inserts a new node on the front of list
def push(self, new_data):

    # 1. Allocates node
    # 2. Put the data in it
    new_node = Node(new_data)

    # 3. Make next of new node as head and
    # previous as None (already None)
    new_node.next = self.head

    # 4. change prev of head node to new_node
    if self.head is not None:
        self.head.prev = new_node

    # 5. move the head to point to the new node
    self.head = new_node

def printList(self, node):
    temp = node
    print "Forward Traversal using next pointer"
    while(node is not None):
        print node.data,
        temp = node
        node = node.next
    print "\nBackward Traversal using prev pointer"
    while(temp):
        print temp.data,
        temp = temp.prev

# Driver program to test the above functions
dll = DoublyLinkedList()
dll.push(5)
dll.push(20);
dll.push(4);
dll.push(3);
dll.push(30)
dll.push(10);
dll.head = dll.mergeSort(dll.head)
print "Linked List after sorting"
dll.printList(dll.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)

```

[Run on IDE](#)

Output:

```

Linked List after sorting
Forward Traversal using next pointer
3 4 5 10 20 30
Backward Traversal using prev pointer
30 20 10 5 4 3

```

Thanks to Goku for providing above implementation in a comment [here](#).

Time Complexity: Time complexity of the above implementation is same as time complexity of [MergeSort for arrays](#). It takes $\Theta(n\log n)$ time.

You may also like to see [QuickSort for doubly linked list](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

[Linked Lists](#) [Sorting](#) [Merge Sort](#)

Related Posts:

- Iteratively Reverse a linked list using only 2 pointers (An Interesting Method)
- Find pair for given sum in a sorted singly linked without extra space
- Find pairs with given sum in doubly linked list
- Unrolled Linked List | Set 1 (Introduction)
- Convert a Binary Tree to a Circular Doubly Link List
- Subtract Two Numbers represented as Linked Lists
- Rearrange a given list such that it consists of alternating minimum maximum elements
- Flatten a multi-level linked list | Set 2 (Depth wise)

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 36 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

HANDBOOK OF ALGORITHMS

Section
Greedy Algorithm

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 1 (Activity Selection Problem)

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems. An optimization problem can be solved using Greedy if the problem has the following property: *At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem.*

If a Greedy Algorithm can solve a problem, then it generally becomes the best method to solve that problem as the Greedy algorithms are in general more efficient than other techniques like Dynamic Programming. But Greedy algorithms cannot always be applied. For example, Fractional Knapsack problem (See [this](#)) can be solved using Greedy, but [0-1 Knapsack](#) cannot be solved using Greedy.

Following are some standard algorithms that are Greedy algorithms.

1) Kruskal's Minimum Spanning Tree (MST): In Kruskal's algorithm, we create a MST by picking edges one by one. The Greedy Choice is to pick the smallest weight edge that doesn't cause a cycle in the MST constructed so far.

2) Prim's Minimum Spanning Tree: In Prim's algorithm also, we create a MST by picking edges one by one. We maintain two sets: set of the vertices already included in MST and the set of the vertices not yet included. The Greedy Choice is to pick the smallest weight edge that connects the two sets.

3) Dijkstra's Shortest Path: The Dijkstra's algorithm is very similar to Prim's algorithm. The shortest path tree is built up, edge by edge. We maintain two sets: set of the vertices already included in the tree and the set of the vertices not yet included. The Greedy Choice is to pick the edge that connects the two sets and is on the smallest weight path from source to the set that contains not yet included vertices.

4) Huffman Coding: Huffman Coding is a loss-less compression technique. It assigns variable length bit codes to different characters. The Greedy Choice is to assign least bit length code to the most frequent character.

The greedy algorithms are sometimes also used to get an approximation for Hard optimization problems. For example, [Traveling Salesman Problem](#) is a NP Hard problem. A Greedy choice for this problem is to pick the nearest unvisited city from the current city at every step. This solutions doesn't always produce the best optimal solution, but can be used to get an approximate optimal solution.

Let us consider the [Activity Selection problem](#) as our first example of Greedy algorithms. Following is the problem statement.

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Example:

Consider the following 6 activities.

```
start[] = {1, 3, 0, 5, 8, 5};
finish[] = {2, 4, 6, 7, 9, 9};
```

The maximum set of activities that can be executed by a single person is {0, 1, 3, 4}

We strongly recommend that you click here and practice it, before moving on to the solution.

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

- 1) Sort the activities according to their finishing time
 - 2) Select the first activity from the sorted array and print it.
 - 3) Do following for remaining activities in the sorted array.
-a) If the start time of this activity is greater than the finish time of previously selected activity then select this activity and print it.

In the following C implementation, it is assumed that the activities are already sorted according to their finish time.

```
#include<stdio.h>

// Prints a maximum set of activities that can be done by a single
// person, one at a time.
// n --> Total number of activities
// s[] --> An array that contains start time of all activities
// f[] --> An array that contains finish time of all activities
void printMaxActivities(int s[], int f[], int n)
{
    int i, j;

    printf ("Following activities are selected \n");

    // The first activity always gets selected
    i = 0;
    printf("%d ", i);

    // Consider rest of the activities
    for (j = 1; j < n; j++)
    {
        // If this activity has start time greater than or
        // equal to the finish time of previously selected
        // activity, then select it
        if (s[j] >= f[i])
        {
            printf ("%d ", j);
            i = j;
        }
    }

    // driver program to test above function
int main()
{
    int s[] = {1, 3, 0, 5, 8, 5};
    int f[] = {2, 4, 6, 7, 9, 9};
    int n = sizeof(s)/sizeof(s[0]);
}
```

```

printMaxActivities(s, f, n);
getchar();
return 0;
}

```

[Run on IDE](#)

Java

```

import java.util.*;
import java.lang.*;
import java.io.*;

class ActivitySelection
{
    // Prints a maximum set of activities that can be done by a single
    // person, one at a time.
    // n --> Total number of activities
    // s[] --> An array that contains start time of all activities
    // f[] --> An array that contains finish time of all activities
    public static void printMaxActivities(int s[], int f[], int n)
    {
        int i, j;

        System.out.print("Following activities are selected : \n");

        // The first activity always gets selected
        i = 0;
        System.out.print(i+" ");

        // Consider rest of the activities
        for (j = 1; j < n; j++)
        {
            // If this activity has start time greater than or
            // equal to the finish time of previously selected
            // activity, then select it
            if (s[j] >= f[i])
            {
                System.out.print(j+" ");
                i = j;
            }
        }

        // driver program to test above function
        public static void main(String[] args)
        {
            int s[] = {1, 3, 0, 5, 8, 5};
            int f[] = {2, 4, 6, 7, 9, 9};
            int n = s.length;

            printMaxActivities(s, f, n);
        }
    }
}

```

[Run on IDE](#)

Python

```

"""The following implementation assumes that the activities
are already sorted according to their finish time"""

"""Prints a maximum set of activities that can be done by a
single person, one at a time"""
# n --> Total number of activities
# s[]--> An array that contains start time of all activities
# f[] --> An array that conatins finish time of all activities

```

```

def printMaxActivities(s , f ):
    n = len(f)
    print "The following activities are selected"

    # The first activity is always selected
    i = 0
    print i,

    # Consider rest of the activities
    for j in xrange(n):

        # If this activity has start time greater than
        # or equal to the finish time of previously
        # selected activity, then select it
        if s[j] >= f[i]:
            print j,
            i = j

# Driver program to test above function
s = [1 , 3 , 0 , 5 , 8 , 5]
f = [2 , 4 , 6 , 7 , 9 , 9]
printMaxActivities(s , f)

# This code is contributed by Nikhil Kumar Singh

```

Run on IDE

Output:

```

Following activities are selected
0 1 3 4

```

How does Greedy Choice work for Activities sorted according to finish time?

Let the give set of activities be $S = \{1, 2, 3, \dots, n\}$ and activities be sorted by finish time. The greedy choice is to always pick activity 1. How come the activity 1 always provides one of the optimal solutions. We can prove it by showing that if there is another solution B with first activity other than 1, then there is also a solution A of same size with activity 1 as first activity. Let the first activity selected by B be k, then there always exist A = $\{B - \{k\}\} \cup \{1\}$. (Note that the activities in B are independent and k has smallest finishing time among all. Since k is not 1, $\text{finish}(k) \geq \text{finish}(1)$).

References:

Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
http://en.wikipedia.org/wiki/Greedy_algorithm

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Greedy](#) [Activity Selection Problem](#) [Greedy](#) [Greedy Algorithm](#)

Related Posts:

- Minimize the sum of product of two arrays with permutations allowed
- Find maximum sum possible equal sum of three stacks
- Minimum sum of two numbers formed from digits of an array
- Maximize array sum after K negations | Set 2
- Minimum edges to reverse to make path from a source to a destination
- Minimum Cost to cut a board into squares
- Maximize array sum after K negations | Set 1
- Job Sequencing Problem | Set 2 (Using Disjoint Set)

(Login to Rate and Mark)

2.2

Average Difficulty : 2.2/5.0
Based on 93 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 2 (Kruskal's Minimum Spanning Tree Algorithm)

What is Minimum Spanning Tree?

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

A minimum spanning tree has $(V - 1)$ edges where V is the number of vertices in the given graph.

What are the applications of Minimum Spanning Tree?

See [this](#) for applications of MST.

Below are the steps for finding MST using Kruskal's algorithm

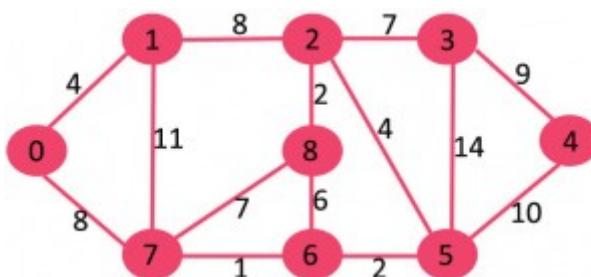
1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are $(V-1)$ edges in the spanning tree.

The step#2 uses [Union-Find algorithm](#) to detect cycle. So we recommend to read following post as a prerequisite.

[Union-Find Algorithm | Set 1 \(Detect Cycle in a Graph\)](#)

[Union-Find Algorithm | Set 2 \(Union By Rank and Path Compression\)](#)

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having $(9 - 1) = 8$ edges.

After sorting:

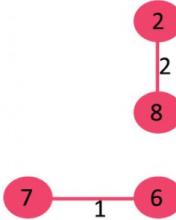
Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from sorted list of edges

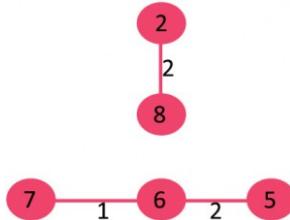
1. *Pick edge 7-6: No cycle is formed, include it.*



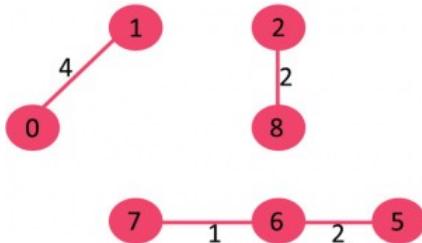
2. *Pick edge 8-2: No cycle is formed, include it.*



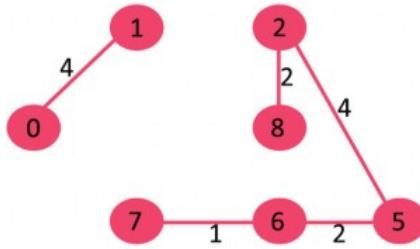
3. *Pick edge 6-5: No cycle is formed, include it.*



4. *Pick edge 0-1: No cycle is formed, include it.*

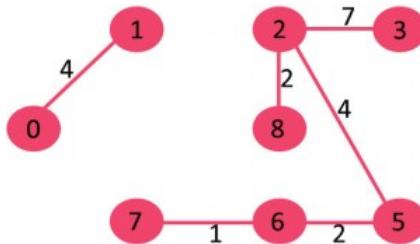


5. *Pick edge 2-5: No cycle is formed, include it.*



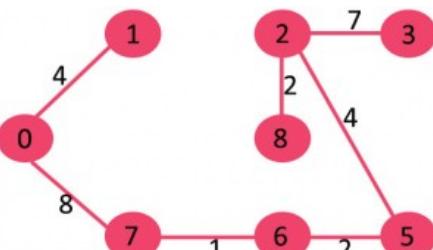
6. Pick edge 8-6: Since including this edge results in cycle, discard it.

7. Pick edge 2-3: No cycle is formed, include it.



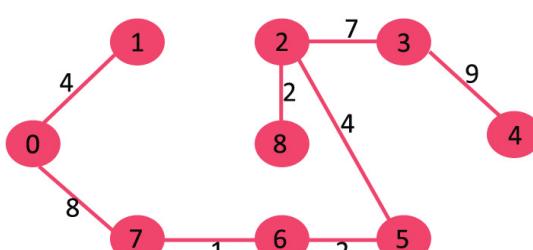
8. Pick edge 7-8: Since including this edge results in cycle, discard it.

9. Pick edge 0-7: No cycle is formed, include it.



10. Pick edge 1-2: Since including this edge results in cycle, discard it.

11. Pick edge 3-4: No cycle is formed, include it.



Since the number of edges included equals $(V - 1)$, the algorithm stops here.

We strongly recommend you to minimize your browser and try this yourself first.

```
// C++ program for Kruskal's algorithm to find Minimum Spanning Tree
// of a given connected, undirected and weighted graph
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// a structure to represent a weighted edge in graph
struct Edge
{
    int src, dest, weight;
};
```

```

// a structure to represent a connected, undirected and weighted graph
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges. Since the graph is
    // undirected, the edge from src to dest is also edge from dest
    // to src. Both are counted as 1 edge here.
    struct Edge* edge;
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );
    return graph;
}

// A structure to represent a subset for union-find
struct subset
{
    int parent;
    int rank;
};

// A utility function to find set of an element i
// (uses path compression technique)
int find(struct subset subsets[], int i)
{
    // find root and make root as parent of i (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high rank tree
    // (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and increment
    // its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Compare two edges according to their weights.
// Used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

```

```

// The main function to construct MST using Kruskal's algorithm
void KruskalMST(struct Graph* graph)
{
    int V = graph->V;
    struct Edge result[V]; // This will store the resultant MST
    int e = 0; // An index variable, used for result[]
    int i = 0; // An index variable, used for sorted edges

    // Step 1: Sort all the edges in non-decreasing order of their weight
    // If we are not allowed to change the given graph, we can create a copy of
    // array of edges
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    // Allocate memory for creating V subsets
    struct subset *subsets =
        (struct subset*) malloc( V * sizeof(struct subset) );

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Number of edges to be taken is equal to V-1
    while (e < V - 1)
    {
        // Step 2: Pick the smallest edge. And increment the index
        // for next iteration
        struct Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        // If including this edge does't cause cycle, include it
        // in result and increment the index of result for next edge
        if (x != y)
        {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
        // Else discard the next_edge
    }

    // print the contents of result[] to display the built MST
    printf("Following are the edges in the constructed MST\n");
    for (i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src, result[i].dest,
                           result[i].weight);
    return;
}

// Driver program to test above functions
int main()
{
    /* Let us create following weighted graph
       10
      0-----1
      | \   |
      6   5 \ 15
      |   \ |
      2-----3
      4     */
    int V = 4; // Number of vertices in graph
    int E = 5; // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    // add edge 0-2
}

```

```

graph->edge[1].src = 0;
graph->edge[1].dest = 2;
graph->edge[1].weight = 6;

// add edge 0-3
graph->edge[2].src = 0;
graph->edge[2].dest = 3;
graph->edge[2].weight = 5;

// add edge 1-3
graph->edge[3].src = 1;
graph->edge[3].dest = 3;
graph->edge[3].weight = 15;

// add edge 2-3
graph->edge[4].src = 2;
graph->edge[4].dest = 3;
graph->edge[4].weight = 4;

KruskalMST(graph);

return 0;
}

```

Run on IDE

Java

```

// Java program for Kruskal's algorithm to find Minimum Spanning Tree
// of a given connected, undirected and weighted graph
import java.util.*;
import java.lang.*;
import java.io.*;

class Graph
{
    // A class to represent a graph edge
    class Edge implements Comparable<Edge>
    {
        int src, dest, weight;

        // Comparator function used for sorting edges based on
        // their weight
        public int compareTo(Edge compareEdge)
        {
            return this.weight - compareEdge.weight;
        }
    };

    // A class to represent a subset for union-find
    class subset
    {
        int parent, rank;
    };

    int V, E;      // V-> no. of vertices & E->no.of edges
    Edge edge[]; // collection of all edges

    // Creates a graph with V vertices and E edges
    Graph(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i=0; i<e; ++i)
            edge[i] = new Edge();
    }

    // A utility function to find set of an element i
    // (uses path compression technique)
    int find(subset subsets[], int i)

```

```

{
    // find root and make root as parent of i (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high rank tree
    // (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and increment
    // its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// The main function to construct MST using Kruskal's algorithm
void KruskalMST()
{
    Edge result[] = new Edge[V]; // This will store the resultant MST
    int e = 0; // An index variable, used for result[]
    int i = 0; // An index variable, used for sorted edges
    for (i=0; i<V; ++i)
        result[i] = new Edge();

    // Step 1: Sort all the edges in non-decreasing order of their
    // weight. If we are not allowed to change the given graph, we
    // can create a copy of array of edges
    Arrays.sort(edge);

    // Allocate memory for creating V subsets
    subset subsets[] = new subset[V];
    for(i=0; i<V; ++i)
        subsets[i]=new subset();

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    i = 0; // Index used to pick next edge

    // Number of edges to be taken is equal to V-1
    while (e < V - 1)
    {
        // Step 2: Pick the smallest edge. And increment the index
        // for next iteration
        Edge next_edge = new Edge();
        next_edge = edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        // If including this edge doesn't cause cycle, include it
        // in result and increment the index of result for next edge
        if (x != y)
        {
    }
}

```

```

        result[e++] = next_edge;
        Union(subsets, x, y);
    }
    // Else discard the next_edge
}

// print the contents of result[] to display the built MST
System.out.println("Following are the edges in the constructed MST");
for (i = 0; i < e; ++i)
    System.out.println(result[i].src+" -- "+result[i].dest+" == "+
                       result[i].weight);
}

// Driver Program
public static void main (String[] args)
{
    /* Let us create following weighted graph
       10
       0-----1
       | \   |
       6 |   5\  | 15
       |   \   |
       2-----3
       4           */
    int V = 4; // Number of vertices in graph
    int E = 5; // Number of edges in graph
    Graph graph = new Graph(V, E);

    // add edge 0-1
    graph.edge[0].src = 0;
    graph.edge[0].dest = 1;
    graph.edge[0].weight = 10;

    // add edge 0-2
    graph.edge[1].src = 0;
    graph.edge[1].dest = 2;
    graph.edge[1].weight = 6;

    // add edge 0-3
    graph.edge[2].src = 0;
    graph.edge[2].dest = 3;
    graph.edge[2].weight = 5;

    // add edge 1-3
    graph.edge[3].src = 1;
    graph.edge[3].dest = 3;
    graph.edge[3].weight = 15;

    // add edge 2-3
    graph.edge[4].src = 2;
    graph.edge[4].dest = 3;
    graph.edge[4].weight = 4;

    graph.KruskalMST();
}
}
//This code is contributed by Aakash Hasija

```

[Run on IDE](#)

Python

```

# Python program for Kruskal's algorithm to find Minimum Spanning Tree
# of a given connected, undirected and weighted graph

from collections import defaultdict

#Class to represent a graph
class Graph:

```

```

def __init__(self,vertices):
    self.V= vertices #No. of vertices
    self.graph = [] # default dictionary to store graph

# function to add an edge to graph
def addEdge(self,u,v,w):
    self.graph.append([u,v,w])

# A utility function to find set of an element i
# (uses path compression technique)
def find(self, parent, i):
    if parent[i] == i:
        return i
    return self.find(parent, parent[i])

# A function that does union of two sets of x and y
# (uses union by rank)
def union(self, parent, rank, x, y):
    xroot = self.find(parent, x)
    yroot = self.find(parent, y)

    # Attach smaller rank tree under root of high rank tree
    # (Union by Rank)
    if rank[xroot] < rank[yroot]:
        parent[xroot] = yroot
    elif rank[xroot] > rank[yroot]:
        parent[yroot] = xroot
    #If ranks are same, then make one as root and increment
    # its rank by one
    else :
        parent[yroot] = xroot
        rank[xroot] += 1

# The main function to construct MST using Kruskal's algorithm
def KruskalMST(self):

    result =[] #This will store the resultant MST

    i = 0 # An index variable, used for sorted edges
    e = 0 # An index variable, used for result[]

    #Step 1: Sort all the edges in non-decreasing order of their
    # weight. If we are not allowed to change the given graph, we
    # can create a copy of graph
    self.graph = sorted(self.graph,key=lambda item: item[2])
    #print self.graph

    parent = [] ; rank = []

    # Create V subsets with single elements
    for node in range(self.V):
        parent.append(node)
        rank.append(0)

    # Number of edges to be taken is equal to V-1
    while e < self.V -1 :

        # Step 2: Pick the smallest edge and increment the index
        # for next iteration
        u,v,w = self.graph[i]
        i = i + 1
        x = self.find(parent, u)
        y = self.find(parent ,v)

        # If including this edge doesn't cause cycle, include it
        # in result and increment the index of result for next edge
        if x != y:
            e = e + 1
            result.append([u,v,w])
            self.union(parent, rank, x, y)
        # Else discard the edge

    # print the contents of result[] to display the built MST

```

```

print "Following are the edges in the constructed MST"
for u,v,weight  in result:
    #print str(u) + " -- " + str(v) + " == " + str(weight)
    print ("%d -- %d == %d" % (u,v,weight))

g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

g.KruskalMST()

#This code is contributed by Neelam Yadav

```

[Run on IDE](#)

Following are the edges in the constructed MST

2 -- 3 == 4
 0 -- 3 == 5
 0 -- 1 == 10

Time Complexity: $O(E\log E)$ or $O(E\log V)$. Sorting of edges takes $O(E\log E)$ time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost $O(\log V)$ time. So overall complexity is $O(E\log E + E\log V)$ time. The value of E can be atmost $O(V^2)$, so $O(\log V)$ are $O(\log E)$ same. Therefore, overall time complexity is $O(E\log E)$ or $O(E\log V)$

References:

<http://www.ics.uci.edu/~eppstein/161/960206.html>
http://en.wikipedia.org/wiki/Minimum_spanning_tree

This article is compiled by [Aashish Barnwal](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

3.5

Average Difficulty : **3.5/5.0**
Based on **73** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 3 (Huffman Coding)

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

The variable-length codes assigned to input characters are [Prefix Codes](#), means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be "cccd" or "ccb" or "acd" or "ab".

See [this](#) for applications of Huffman Coding.

There are mainly two major parts in Huffman Coding

- 1) Build a Huffman Tree from input characters.
- 2) Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps #2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

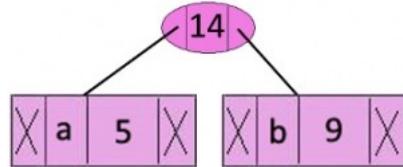
Let us understand the algorithm with an example:

character	Frequency
a	5
b	9
c	12
d	13

e	16
f	45

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

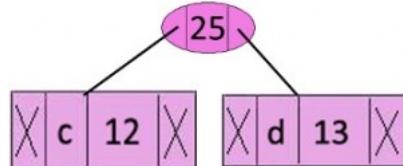
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency $5 + 9 = 14$.



Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

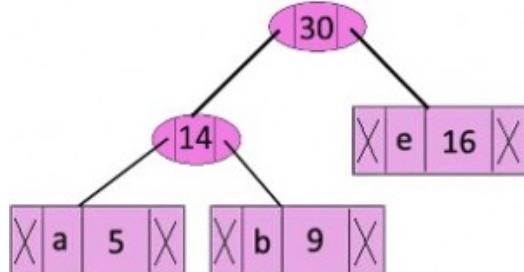
Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency $12 + 13 = 25$



Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes.

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

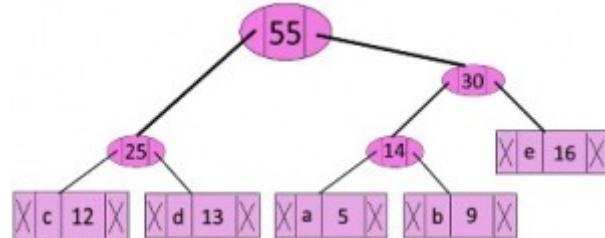
Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency $14 + 16 = 30$



Now min heap contains 3 nodes.

character	Frequency
Internal Node	25
Internal Node	30
f	45

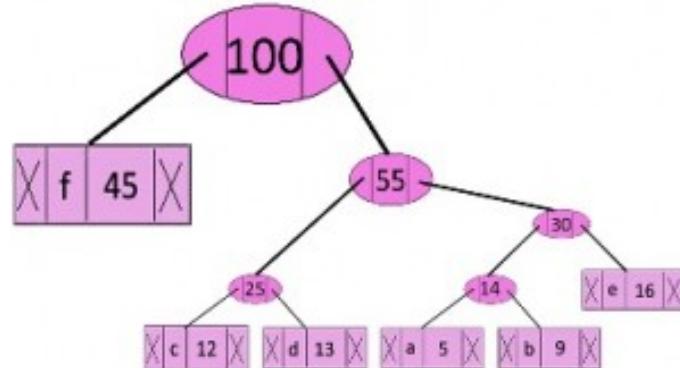
Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency $25 + 30 = 55$



Now min heap contains 2 nodes.

character	Frequency
f	45
Internal Node	55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency $45 + 55 = 100$



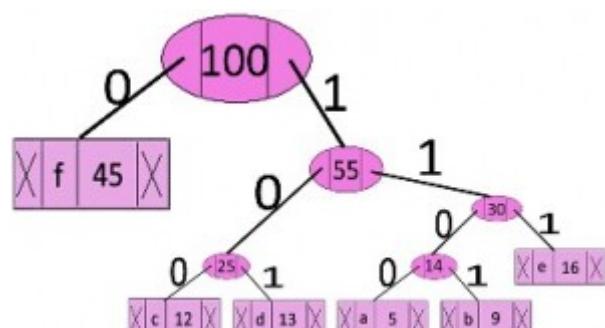
Now min heap contains only one node.

character	Frequency
Internal Node	100

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111

```
// C program for Huffman Coding
#include <stdio.h>
#include <stdlib.h>

// This constant can be avoided by explicitly calculating height of Huffman Tree
#define MAX_TREE_HT 100

// A Huffman tree node
struct MinHeapNode
{
    char data; // One of the input characters
    unsigned freq; // Frequency of the character
    struct MinHeapNode *left, *right; // Left and right child of this node
};

// A Min Heap: Collection of min heap (or Huffman tree) nodes
struct MinHeap
{
    unsigned size; // Current size of min heap
    unsigned capacity; // capacity of min heap
    struct MinHeapNode **array; // Array of minheap node pointers
};

// A utility function allocate a new min heap node with given character
// and frequency of the character
struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp =
        (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

// A utility function to create a min heap of given capacity
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap =
        (struct MinHeap*) malloc(sizeof(struct MinHeap));
    minHeap->size = 0; // current size is 0
    minHeap->capacity = capacity;
    minHeap->array =
        (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}

// A utility function to swap two min heap nodes
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
```

```

// The standard minHeapify function.
void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;

    if (smallest != idx)
    {
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

// A utility function to check if size of heap is 1 or not
int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

// A standard function to extract minimum value node from heap
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

// A utility function to insert a new node to Min Heap
void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode)
{
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1)/2]->freq)
    {
        minHeap->array[i] = minHeap->array[(i - 1)/2];
        i = (i - 1)/2;
    }
    minHeap->array[i] = minHeapNode;
}

// A standard function to build min heap
void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

// A utility function to print an array of size n
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}

// Utility function to check if this node is leaf
int isLeaf(struct MinHeapNode* root)
{
    return !(root->left) && !(root->right) ;
}

```

```

// Creates a min heap of capacity equal to size and inserts all character of
// data[] in min heap. Initially size of min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

// The main function that builds Huffman tree
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

    // Step 1: Create a min heap of capacity equal to size. Initially, there are
    // modes equal to size.
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);

    // Iterate while size of heap doesn't become 1
    while (!isSizeOne(minHeap))
    {
        // Step 2: Extract the two minimum freq items from min heap
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        // Step 3: Create a new internal node with frequency equal to the
        // sum of the two nodes frequencies. Make the two extracted node as
        // left and right children of this new node. Add this node to the min heap
        // '$' is a special value for internal nodes, not used
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }

    // Step 4: The remaining node is the root node and the tree is complete.
    return extractMin(minHeap);
}

// Prints huffman codes from the root of Huffman Tree. It uses arr[] to
// store codes
void printCodes(struct MinHeapNode* root, int arr[], int top)
{
    // Assign 0 to left edge and recur
    if (root->left)
    {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    // Assign 1 to right edge and recur
    if (root->right)
    {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    // If this is a leaf node, then it contains one of the input
    // characters, print the character and its code from arr[]
    if (isLeaf(root))
    {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

// The main function that builds a Huffman Tree and print codes by traversing
// the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size)
{
}

```

```

// Construct Huffman Tree
struct MinHeapNode* root = buildHuffmanTree(data, freq, size);

// Print Huffman codes using the Huffman tree built above
int arr[MAX_TREE_HT], top = 0;
printCodes(root, arr, top);
}

// Driver program to test above functions
int main()
{
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr)/sizeof(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}

```

Run on IDE

C++ using STL

```

// C++ program for Huffman Coding
#include <bits/stdc++.h>
using namespace std;

// A Huffman tree node
struct MinHeapNode
{
    char data; // One of the input characters
    unsigned freq; // Frequency of the character
    MinHeapNode *left, *right; // Left and right child

    MinHeapNode(char data, unsigned freq)
    {
        left = right = NULL;
        this->data = data;
        this->freq = freq;
    }
};

// For comparison of two heap nodes (needed in min heap)
struct compare
{
    bool operator()(MinHeapNode* l, MinHeapNode* r)
    {
        return (l->freq > r->freq);
    }
};

// Prints huffman codes from the root of Huffman Tree.
void printCodes(struct MinHeapNode* root, string str)
{
    if (!root)
        return;

    if (root->data != '$')
        cout << root->data << ":" << str << "\n";

    printCodes(root->left, str + "0");
    printCodes(root->right, str + "1");
}

// The main function that builds a Huffman Tree and
// print codes by traversing the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

    // Create a min heap & inserts all characters of data[]
    priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;

```

```

for (int i = 0; i < size; ++i)
    minHeap.push(new MinHeapNode(data[i], freq[i]));

// Iterate while size of heap doesn't become 1
while (minHeap.size() != 1)
{
    // Extract the two minimum freq items from min heap
    left = minHeap.top();
    minHeap.pop();

    right = minHeap.top();
    minHeap.pop();

    // Create a new internal node with frequency equal to the
    // sum of the two nodes frequencies. Make the two extracted
    // node as left and right children of this new node. Add
    // this node to the min heap
    // '$' is a special value for internal nodes, not used
    top = new MinHeapNode('$', left->freq + right->freq);
    top->left = left;
    top->right = right;
    minHeap.push(top);
}

// Print Huffman codes using the Huffman tree built above
printCodes(minHeap.top(), "");
}

// Driver program to test above functions
int main()
{
    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };
    int size = sizeof(arr) / sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;
}
// This code is contributed by Aditya Goel

```

[Run on IDE](#)

f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111

Time complexity: O(nlogn) where n is the number of unique characters. If there are n nodes, extractMin() is called 2*(n - 1) times. extractMin() takes O(logn) time as it calls minHeapify(). So, overall complexity is O(nlogn).

If the input array is sorted, there exists a linear time algorithm. We will soon be discussing in our next post.

Reference:

http://en.wikipedia.org/wiki/Huffman_coding

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Greedy](#) [Greedy Algorithm](#) [Huffman Coding](#)

Related Posts:

- Minimize the sum of product of two arrays with permutations allowed
- Find maximum sum possible equal sum of three stacks
- Minimum sum of two numbers formed from digits of an array
- Maximize array sum after K negations | Set 2
- Minimum edges to reverse to make path from a source to a destination
- Minimum Cost to cut a board into squares
- Maximize array sum after K negations | Set 1
- Job Sequencing Problem | Set 2 (Using Disjoint Set)

(Login to Rate and Mark)

4.2

Average Difficulty : 4.2/5.0
Based on 39 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 4 (Efficient Huffman Coding for Sorted Input)

We recommend to read following post as a prerequisite for this.

[Greedy Algorithms | Set 3 \(Huffman Coding\)](#)

Time complexity of the algorithm discussed in above post is $O(n\log n)$. If we know that the given array is sorted (by non-decreasing order of frequency), we can generate Huffman codes in $O(n)$ time. Following is a $O(n)$ algorithm for sorted input.

1. Create two empty queues.
2. Create a leaf node for each unique character and Enqueue it to the first queue in non-decreasing order of frequency. Initially second queue is empty.
3. Dequeue two nodes with the minimum frequency by examining the front of both queues. Repeat following steps two times
 -a) If second queue is empty, dequeue from first queue.
 -b) If first queue is empty, dequeue from second queue.
 -c) Else, compare the front of two queues and dequeue the minimum.
4. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first Dequeued node as its left child and the second Dequeued node as right child. Enqueue this node to second queue.
5. Repeat steps #3 and #4 until there is more than one node in the queues. The remaining node is the root node and the tree is complete.

```
// C Program for Efficient Huffman Coding for Sorted input
#include <stdio.h>
#include <stdlib.h>

// This constant can be avoided by explicitly calculating height of Huffman Tree
#define MAX_TREE_HT 100

// A node of huffman tree
struct QueueNode
{
    char data;
    unsigned freq;
    struct QueueNode *left, *right;
};

// Structure for Queue: collection of Huffman Tree nodes (or QueueNodes)
struct Queue
{
    int front, rear;
    int capacity;
}
```

```

struct QueueNode **array;
};

// A utility function to create a new QueueNode
struct QueueNode* newNode(char data, unsigned freq)
{
    struct QueueNode* temp =
        (struct QueueNode*) malloc(sizeof(struct QueueNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

// A utility function to create a Queue of given capacity
struct Queue* createQueue(int capacity)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue->front = queue->rear = -1;
    queue->capacity = capacity;
    queue->array =
        (struct QueueNode**) malloc(queue->capacity * sizeof(struct QueueNode*));
    return queue;
}

// A utility function to check if size of given queue is 1
int isSizeOne(struct Queue* queue)
{
    return queue->front == queue->rear && queue->front != -1;
}

// A utility function to check if given queue is empty
int isEmpty(struct Queue* queue)
{
    return queue->front == -1;
}

// A utility function to check if given queue is full
int isFull(struct Queue* queue)
{
    return queue->rear == queue->capacity - 1;
}

// A utility function to add an item to queue
void enQueue(struct Queue* queue, struct QueueNode* item)
{
    if (isFull(queue))
        return;
    queue->array[++queue->rear] = item;
    if (queue->front == -1)
        ++queue->front;
}

// A utility function to remove an item from queue
struct QueueNode* deQueue(struct Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
    struct QueueNode* temp = queue->array[queue->front];
    if (queue->front == queue->rear) // If there is only one item in queue
        queue->front = queue->rear = -1;
    else
        ++queue->front;
    return temp;
}

// A utility function to get front of queue
struct QueueNode* getFront(struct Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
    return queue->array[queue->front];
}

```

```

/* A function to get minimum item from two queues */
struct QueueNode* findMin(struct Queue* firstQueue, struct Queue* secondQueue)
{
    // Step 3.a: If second queue is empty, dequeue from first queue
    if (isEmpty(firstQueue))
        return deQueue(secondQueue);

    // Step 3.b: If first queue is empty, dequeue from second queue
    if (isEmpty(secondQueue))
        return deQueue(firstQueue);

    // Step 3.c: Else, compare the front of two queues and dequeue minimum
    if (getFront(firstQueue)->freq < getFront(secondQueue)->freq)
        return deQueue(firstQueue);

    return deQueue(secondQueue);
}

// Utility function to check if this node is leaf
int isLeaf(struct QueueNode* root)
{
    return !(root->left) && !(root->right) ;
}

// A utility function to print an array of size n
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}

// The main function that builds Huffman tree
struct QueueNode* buildHuffmanTree(char data[], int freq[], int size)
{
    struct QueueNode *left, *right, *top;

    // Step 1: Create two empty queues
    struct Queue* firstQueue = createQueue(size);
    struct Queue* secondQueue = createQueue(size);

    // Step 2: Create a leaf node for each unique character and Enqueue it to
    // the first queue in non-decreasing order of frequency. Initially second
    // queue is empty
    for (int i = 0; i < size; ++i)
        enQueue(firstQueue, newNode(data[i], freq[i]));

    // Run while Queues contain more than one node. Finally, first queue will
    // be empty and second queue will contain only one node
    while (!(isEmpty(firstQueue) && isSizeOne(secondQueue)))
    {
        // Step 3: Dequeue two nodes with the minimum frequency by examining
        // the front of both queues
        left = findMin(firstQueue, secondQueue);
        right = findMin(firstQueue, secondQueue);

        // Step 4: Create a new internal node with frequency equal to the sum
        // of the two nodes frequencies. Enqueue this node to second queue.
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        enQueue(secondQueue, top);
    }

    return deQueue(secondQueue);
}

// Prints huffman codes from the root of Huffman Tree. It uses arr[] to
// store codes
void printCodes(struct QueueNode* root, int arr[], int top)
{
    // Assign 0 to left edge and recur
    if (root->left)

```

```

{
    arr[top] = 0;
    printCodes(root->left, arr, top + 1);
}

// Assign 1 to right edge and recur
if (root->right)
{
    arr[top] = 1;
    printCodes(root->right, arr, top + 1);
}

// If this is a leaf node, then it contains one of the input
// characters, print the character and its code from arr[]
if (isLeaf(root))
{
    printf("%c: ", root->data);
    printArr(arr, top);
}
}

// The main function that builds a Huffman Tree and print codes by traversing
// the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size)
{
    // Construct Huffman Tree
    struct QueueNode* root = buildHuffmanTree(data, freq, size);

    // Print Huffman codes using the Huffman tree built above
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}

// Driver program to test above functions
int main()
{
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr)/sizeof(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}

```

Run on IDE

Output:

```

f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111

```

Time complexity: O(n)

If the input is not sorted, it need to be sorted first before it can be processed by the above algorithm. Sorting can be done using heap-sort or merge-sort both of which run in Theta(nlogn). So, the overall time complexity becomes O(nlogn) for unsorted input.

Reference:

http://en.wikipedia.org/wiki/Huffman_coding

This article is compiled by [Aashish Barnwal](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Greedy](#) [Greedy Algorithm](#) [Huffman Coding](#)

Related Posts:

- Minimize the sum of product of two arrays with permutations allowed
- Find maximum sum possible equal sum of three stacks
- Minimum sum of two numbers formed from digits of an array
- Maximize array sum after K negations | Set 2
- Minimum edges to reverse to make path from a source to a destination
- Minimum Cost to cut a board into squares
- Maximize array sum after K negations | Set 1
- Job Sequencing Problem | Set 2 (Using Disjoint Set)

(Login to Rate and Mark)

2.9

Average Difficulty : **2.9/5.0**
Based on **12** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 5 (Prim's Minimum Spanning Tree (MST))

We have discussed [Kruskal's algorithm for Minimum Spanning Tree](#). Like Kruskal's algorithm, Prim's algorithm is also a [Greedy algorithm](#). It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

A group of edges that connects two set of vertices in a graph is called [cut in graph theory](#). So, *at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices)*.

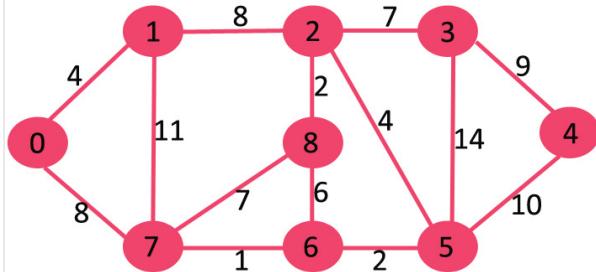
How does Prim's Algorithm Work? The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning Tree*. And they must be connected with the minimum weight edge to make it a *Minimum Spanning Tree*.

Algorithm

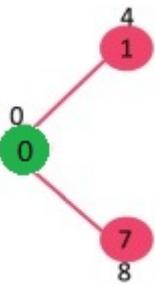
- 1) Create a set *mstSet* that keeps track of vertices already included in MST.
- 2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
- 3) While *mstSet* doesn't include all vertices
 -a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
 -b) Include *u* to *mstSet*.
 -c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*

The idea of using key values is to pick the minimum weight edge from [cut](#). The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

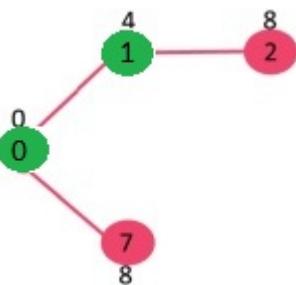
Let us understand with the following example:



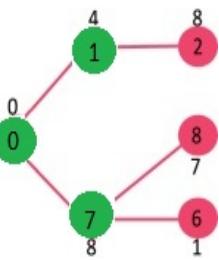
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet*, update key values of adjacent vertices. Adjacent vertices of 0 are 1 and 7. The key values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



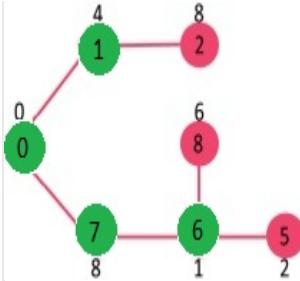
Pick the vertex with minimum key value and not already included in MST (not in *mstSET*). The vertex 1 is picked and added to *mstSet*. So *mstSet* now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



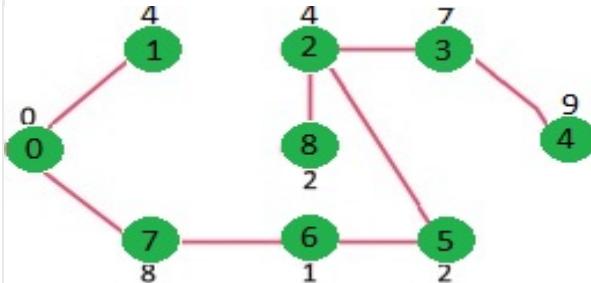
Pick the vertex with minimum key value and not already included in MST (not in *mstSET*). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So *mstSet* now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (7 and 1 respectively).



Pick the vertex with minimum key value and not already included in MST (not in *mstSET*). Vertex 6 is picked. So *mstSet* now becomes {0, 1, 7, 6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



We repeat the above steps until *mstSet* includes all vertices of given graph. Finally, we get the following graph.



We strongly recommend that you click here and practice it, before moving on to the solution.

How to implement the above algorithm?

We use a boolean array *mstSet[]* to represent the set of vertices included in MST. If a value *mstSet[v]* is true, then vertex *v* is included in MST, otherwise not. Array *key[]* is used to store key values of all vertices. Another array *parent[]* to store indexes of parent nodes in MST. The parent array is the output array which is used to show the constructed MST.

```
// A C / C++ program for Prim's Minimum Spanning Tree (MST) algorithm.
// The program is for adjacency matrix representation of the graph
```

```
#include <stdio.h>
#include <limits.h>

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with minimum key value, from
// the set of vertices not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed MST stored in parent[]
int printMST(int parent[], int n, int graph[V][V])
{
    printf("Edge   Weight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d   %d \n", parent[i], i, graph[i][parent[i]]);
```

```

}

// Function to construct and print MST for a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    int parent[V]; // Array to store constructed MST
    int key[V]; // Key values used to pick minimum weight edge in cut
    bool mstSet[V]; // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V-1; count++)
    {
        // Pick the minimum key vertex from the set of vertices
        // not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of the adjacent vertices of m
        // mstSet[v] is false for vertices not yet included in MST
        // Update the key only if graph[u][v] is smaller than key[v]
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // print the constructed MST
    printMST(parent, V, graph);
}

```

```

// driver program to test above function
int main()
{
    /* Let us create the following graph
       2   3
     (0)--(1)--(2)
      |   / \ |
      6| 8/   \5 |7
      |   / \ |
     (3)-----(4)
          9   */
    int graph[V][V] = {{0, 2, 0, 6, 0},
                       {2, 0, 3, 8, 5},
                       {0, 3, 0, 0, 7},
                       {6, 8, 0, 0, 9},
                       {0, 5, 7, 9, 0},
                      };

    // Print the solution
    primMST(graph);

    return 0;
}

```

Run on IDE

Java

```
// A Java program for Prim's Minimum Spanning Tree (MST) algorithm.
// The program is for adjacency matrix representation of the graph

import java.util.*;
import java.lang.*;
import java.io.*;

class MST
{
    // Number of vertices in the graph
    private static final int V=5;

    // A utility function to find the vertex with minimum key
    // value, from the set of vertices not yet included in MST
    int minKey(int key[], Boolean mstSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index=-1;

        for (int v = 0; v < V; v++)
            if (mstSet[v] == false && key[v] < min)
            {
                min = key[v];
                min_index = v;
            }

        return min_index;
    }

    // A utility function to print the constructed MST stored in
    // parent[]
    void printMST(int parent[], int n, int graph[][][])
    {
        System.out.println("Edge   Weight");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i]+ " - " + i+ "   " +
                               graph[i][parent[i]]);
    }

    // Function to construct and print MST for a graph represented
    // using adjacency matrix representation
    void primMST(int graph[][][])
    {
        // Array to store constructed MST
        int parent[] = new int[V];

        // Key values used to pick minimum weight edge in cut
        int key[] = new int [V];

        // To represent set of vertices not yet included in MST
        Boolean mstSet[] = new Boolean[V];

        // Initialize all keys as INFINITE
        for (int i = 0; i < V; i++)
        {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        // Always include first 1st vertex in MST.
        key[0] = 0;      // Make key 0 so that this vertex is
                        // picked as first vertex
        parent[0] = -1; // First node is always root of MST

        // The MST will have V vertices
        for (int count = 0; count < V-1; count++)
        {
            // Pick thd minimum key vertex from the set of vertices
            // not yet included in MST
            int u = minKey(key, mstSet);

```

```

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of the adjacent
        // vertices of the picked vertex. Consider only those
        // vertices which are not yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent vertices of m
            // mstSet[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] != 0 && mstSet[v] == false &&
                graph[u][v] < key[v])
            {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }

        // print the constructed MST
        printMST(parent, V, graph);
    }

    public static void main (String[] args)
    {
        /* Let us create the following graph
           2   3
         (0)--(1)--(2)
          |   / \ |
          6| 8/   \5 |7
          | /     \ |
        (3)-----(4)
          9
        */
        MST t = new MST();
        int graph[][][] = new int[][][] {{ {0, 2, 0, 6, 0}, {2, 0, 3, 8, 5}, {0, 3, 0, 0, 7}, {6, 8, 0, 0, 9}, {0, 5, 7, 9, 0} }, { {0, 1, 2, 3, 4}, {1, 0, 2, 3, 4}, {2, 0, 1, 3, 4}, {3, 0, 1, 2, 4}, {4, 0, 1, 2, 3} } };
        // Print the solution
        t.primMST(graph);
    }
}
// This code is contributed by Aakash Hasija

```

Run on IDE

Output:

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

Time Complexity of the above program is $O(V^2)$. If the input graph is represented using adjacency list, then the time complexity of Prim's algorithm can be reduced to $O(E \log V)$ with the help of binary heap. Please see [Prim's MST for Adjacency List Representation](#) for more details.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

[Graph](#) [Greedy](#) [Graph](#) [Greedy](#) [Greedy Algorithm](#) [Minimum Spanning Tree](#) [Prim's Algorithm.MST](#)

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

2.7

Average Difficulty : 2.7/5.0
Based on 47 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 6 (Prim's MST for Adjacency List Representation)

We recommend to read following two posts as a prerequisite of this post.

1. [Greedy Algorithms | Set 5 \(Prim's Minimum Spanning Tree \(MST\)\)](#)
2. [Graph and its representations](#)

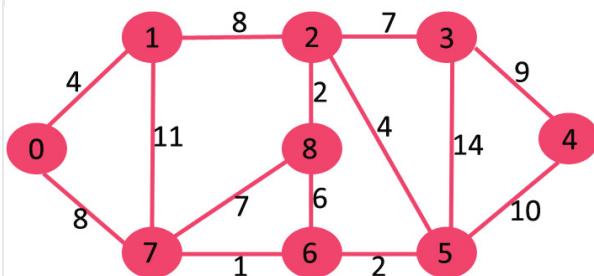
We have discussed Prim's algorithm and its implementation for adjacency matrix representation of graphs. The time complexity for the matrix representation is $O(V^2)$. In this post, $O(E \log V)$ algorithm for adjacency list representation is discussed.

As discussed in the previous post, in Prim's algorithm, two sets are maintained, one set contains list of vertices already included in MST, other set contains vertices not yet included. With adjacency list representation, all vertices of a graph can be traversed in $O(V+E)$ time using BFS. The idea is to traverse all vertices of graph using BFS and use a Min Heap to store the vertices not yet included in MST. Min Heap is used as a priority queue to get the minimum weight edge from the cut. Min Heap is used as time complexity of operations like extracting minimum element and decreasing key value is $O(\log V)$ in Min Heap.

Following are the detailed steps.

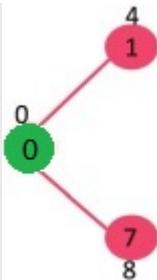
- 1) Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and key value of the vertex.
- 2) Initialize Min Heap with first vertex as root (the key value assigned to first vertex is 0). The key value assigned to all other vertices is INF (infinite).
- 3) While Min Heap is not empty, do following
 - a) Extract the min value node from Min Heap. Let the extracted vertex be u .
 - b) For every adjacent vertex v of u , check if v is in Min Heap (not yet included in MST). If v is in Min Heap and its key value is more than weight of $u-v$, then update the key value of v as weight of $u-v$.

Let us understand the above algorithm with the following example:

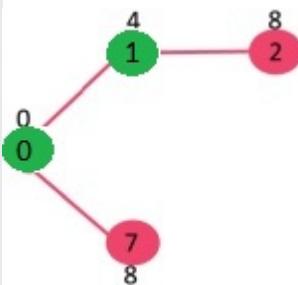


Initially, key value of first vertex is 0 and INF (infinite) for all other vertices. So vertex 0 is extracted from Min Heap and key values of vertices adjacent to 0 (1 and 7) are updated. Min Heap contains all vertices except vertex 0.

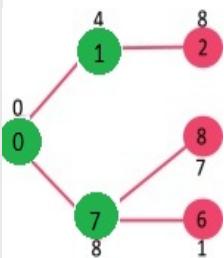
The vertices in green color are the vertices included in MST.



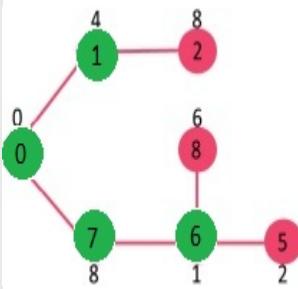
Since key value of vertex 1 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 1 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 1 to the adjacent). Min Heap contains all vertices except vertex 0 and 1.



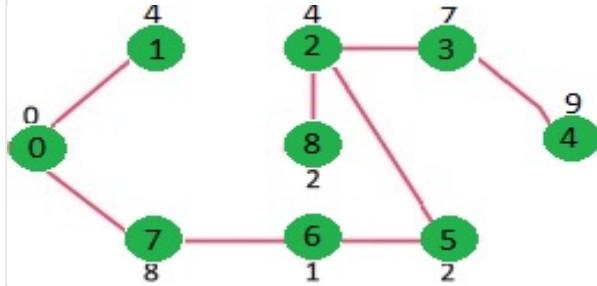
Since key value of vertex 7 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 7 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 7 to the adjacent). Min Heap contains all vertices except vertex 0, 1 and 7.



Since key value of vertex 6 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 6 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 6 to the adjacent). Min Heap contains all vertices except vertex 0, 1, 7 and 6.



The above steps are repeated for rest of the nodes in Min Heap till Min Heap becomes empty



```
// C / C++ program for Prim's MST for adjacency list representation of graph
```

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// A structure to represent a node in adjacency list
struct AdjListNode
{
    int dest;
    int weight;
    struct AdjListNode* next;
};

// A structure to represent an adjacency list
struct AdjList
{
    struct AdjListNode *head; // pointer to head node of list
};

// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph
{
    int V;
    struct AdjList* array;
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest, int weight)
{
    struct AdjListNode* newNode =
        (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = NULL;
    return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;

    // Create an array of adjacency lists. Size of array will be V
    graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));

    // Initialize each adjacency list as empty by making head as NULL
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest, int weight)
{
    // Add an edge from src to dest. A new node is added to the adjacency
    // list of src. The node is added at the begining
    struct AdjListNode* newNode = newAdjListNode(dest, weight);
}
```

```

newNode->next = graph->array[src].head;
graph->array[src].head = newNode;

// Since graph is undirected, add an edge from dest to src also
newNode = newAdjListNode(src, weight);
newNode->next = graph->array[dest].head;
graph->array[dest].head = newNode;
}

// Structure to represent a min heap node
struct MinHeapNode
{
    int v;
    int key;
};

// Structure to represent a min heap
struct MinHeap
{
    int size;      // Number of heap nodes present currently
    int capacity; // Capacity of min heap
    int *pos;      // This is needed for decreaseKey()
    struct MinHeapNode **array;
};

// A utility function to create a new Min Heap Node
struct MinHeapNode* newMinHeapNode(int v, int key)
{
    struct MinHeapNode* minHeapNode =
        (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
    minHeapNode->v = v;
    minHeapNode->key = key;
    return minHeapNode;
}

// A utilit function to create a Min Heap
struct MinHeap* createMinHeap(int capacity)
{
    struct MinHeap* minHeap =
        (struct MinHeap*) malloc(sizeof(struct MinHeap));
    minHeap->pos = (int*) malloc(capacity * sizeof(int));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array =
        (struct MinHeapNode**) malloc(capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}

// A utility function to swap two nodes of min heap. Needed for min heapify
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// A standard function to heapify at given idx
// This function also updates position of nodes when they are swapped.
// Position is needed for decreaseKey()
void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest, left, right;
    smallest = idx;
    left = 2 * idx + 1;
    right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->array[left]->key < minHeap->array[smallest]->key )
        smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->key < minHeap->array[smallest]->key )
        smallest = right;
}

```

```

if (smallest != idx)
{
    // The nodes to be swapped in min heap
    MinHeapNode *smallestNode = minHeap->array[smallest];
    MinHeapNode *idxNode = minHeap->array[idx];

    // Swap positions
    minHeap->pos[smallestNode->v] = idx;
    minHeap->pos[idxNode->v] = smallest;

    // Swap nodes
    swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);

    minHeapify(minHeap, smallest);
}

// A utility function to check if the given minHeap is ampty or not
int isEmpty(struct MinHeap* minHeap)
{
    return minHeap->size == 0;
}

// Standard function to extract minimum node from heap
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    if (isEmpty(minHeap))
        return NULL;

    // Store the root node
    struct MinHeapNode* root = minHeap->array[0];

    // Replace root node with last node
    struct MinHeapNode* lastNode = minHeap->array[minHeap->size - 1];
    minHeap->array[0] = lastNode;

    // Update position of last node
    minHeap->pos[root->v] = minHeap->size-1;
    minHeap->pos[lastNode->v] = 0;

    // Reduce heap size and heapify root
    --minHeap->size;
    minHeapify(minHeap, 0);

    return root;
}

// Function to decreasy key value of a given vertex v. This function
// uses pos[] of min heap to get the current index of node in min heap
void decreaseKey(struct MinHeap* minHeap, int v, int key)
{
    // Get the index of v in heap array
    int i = minHeap->pos[v];

    // Get the node and update its key value
    minHeap->array[i]->key = key;

    // Travel up while the complete tree is not hepified.
    // This is a O(Logn) loop
    while (i && minHeap->array[i]->key < minHeap->array[(i - 1) / 2]->key)
    {
        // Swap this node with its parent
        minHeap->pos[minHeap->array[i]->v] = (i-1)/2;
        minHeap->pos[minHeap->array[(i-1)/2]->v] = i;
        swapMinHeapNode(&minHeap->array[i], &minHeap->array[(i - 1) / 2]);

        // move to parent index
        i = (i - 1) / 2;
    }

    // A utility function to check if a given vertex
    // 'v' is in min heap or not
    bool isInMinHeap(struct MinHeap *minHeap, int v)
}

```

```

{
    if (minHeap->pos[v] < minHeap->size)
        return true;
    return false;
}

// A utility function used to print the constructed MST
void printArr(int arr[], int n)
{
    for (int i = 1; i < n; ++i)
        printf("%d - %d\n", arr[i], i);
}

// The main function that constructs Minimum Spanning Tree (MST)
// using Prim's algorithm
void PrimMST(struct Graph* graph)
{
    int V = graph->V; // Get the number of vertices in graph
    int parent[V]; // Array to store constructed MST
    int key[V]; // Key values used to pick minimum weight edge in cut

    // minHeap represents set E
    struct MinHeap* minHeap = createMinHeap(V);

    // Initialize min heap with all vertices. Key value of
    // all vertices (except 0th vertex) is initially infinite
    for (int v = 1; v < V; ++v)
    {
        parent[v] = -1;
        key[v] = INT_MAX;
        minHeap->array[v] = newMinHeapNode(v, key[v]);
        minHeap->pos[v] = v;
    }

    // Make key value of 0th vertex as 0 so that it
    // is extracted first
    key[0] = 0;
    minHeap->array[0] = newMinHeapNode(0, key[0]);
    minHeap->pos[0] = 0;

    // Initially size of min heap is equal to V
    minHeap->size = V;

    // In the followin loop, min heap contains all nodes
    // not yet added to MST.
    while (!isEmpty(minHeap))
    {
        // Extract the vertex with minimum key value
        struct MinHeapNode* minHeapNode = extractMin(minHeap);
        int u = minHeapNode->v; // Store the extracted vertex number

        // Traverse through all adjacent vertices of u (the extracted
        // vertex) and update their key values
        struct AdjListNode* pCrawl = graph->array[u].head;
        while (pCrawl != NULL)
        {
            int v = pCrawl->dest;

            // If v is not yet included in MST and weight of u-v is
            // less than key value of v, then update key value and
            // parent of v
            if (isInMinHeap(minHeap, v) && pCrawl->weight < key[v])
            {
                key[v] = pCrawl->weight;
                parent[v] = u;
                decreaseKey(minHeap, v, key[v]);
            }
            pCrawl = pCrawl->next;
        }
    }

    // print edges of MST
    printArr(parent, V);
}

```

```
// Driver program to test above functions
int main()
{
    // Let us create the graph given in above fugure
    int V = 9;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 4);
    addEdge(graph, 0, 7, 8);
    addEdge(graph, 1, 2, 8);
    addEdge(graph, 1, 7, 11);
    addEdge(graph, 2, 3, 7);
    addEdge(graph, 2, 8, 2);
    addEdge(graph, 2, 5, 4);
    addEdge(graph, 3, 4, 9);
    addEdge(graph, 3, 5, 14);
    addEdge(graph, 4, 5, 10);
    addEdge(graph, 5, 6, 2);
    addEdge(graph, 6, 7, 1);
    addEdge(graph, 6, 8, 6);
    addEdge(graph, 7, 8, 7);

    PrimMST(graph);

    return 0;
}
```

[Run on IDE](#)

Output:

```
0 - 1
5 - 2
2 - 3
3 - 4
6 - 5
7 - 6
0 - 7
2 - 8
```

Time Complexity: The time complexity of the above code/algorithm looks $O(V^2)$ as there are two nested while loops. If we take a closer look, we can observe that the statements in inner loop are executed $O(V+E)$ times (similar to BFS). The inner loop has decreaseKey() operation which takes $O(\log V)$ time. So overall time complexity is $O(E+V)*O(\log V)$ which is $O((E+V)*\log V) = O(E \log V)$ (For a connected graph, $V = O(E)$)

References:

Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

http://en.wikipedia.org/wiki/Prim's_algorithm

This article is compiled by [Aashish Barnwal](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Graph Greedy Graph Greedy Algorithm MST

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

4.5

Average Difficulty : 4.5/5.0
Based on 24 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm)

Given a graph and a source vertex in graph, find shortest paths from source to all vertices in the given graph.

Dijkstra's algorithm is very similar to [Prim's algorithm for minimum spanning tree](#). Like Prim's MST, we generate a *SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source.

Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

Algorithm

1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

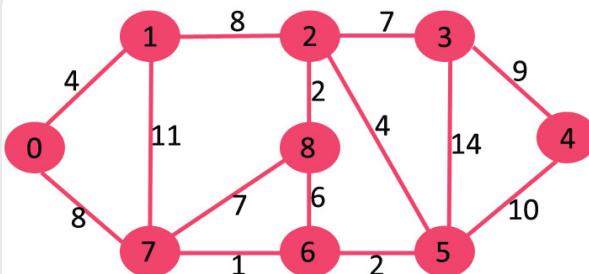
3) While *sptSet* doesn't include all vertices

....a) Pick a vertex *u* which is not there in *sptSet* and has minimum distance value.

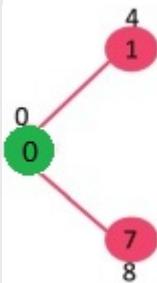
....b) Include *u* to *sptSet*.

....c) Update distance value of all adjacent vertices of *u*. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex *v*, if sum of distance value of *u* (from source) and weight of edge *u-v*, is less than the distance value of *v*, then update the distance value of *v*.

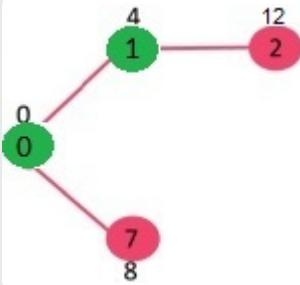
Let us understand with the following example:



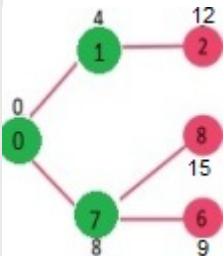
The set *sptSet* is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green color.



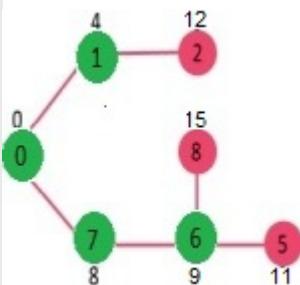
Pick the vertex with minimum distance value and not already included in SPT (not in *sptSet*). The vertex 1 is picked and added to *sptSet*. So *sptSet* now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



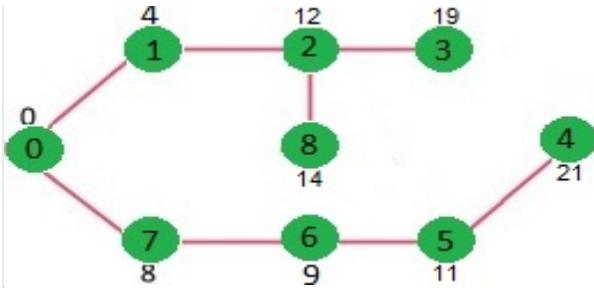
Pick the vertex with minimum distance value and not already included in SPT (not in *sptSet*). Vertex 7 is picked. So *sptSet* now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance value and not already included in SPT (not in *sptSet*). Vertex 6 is picked. So *sptSet* now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until *sptSet* doesn't include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



How to implement the above algorithm?

We strongly recommend that you click here and practice it, before moving on to the solution.

We use a boolean array `sptSet[]` to represent the set of vertices included in SPT. If a value `sptSet[v]` is true, then vertex `v` is included in SPT, otherwise not. Array `dist[]` is used to store shortest distance values of all vertices.

```
// A C / C++ program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

#include <stdio.h>
#include <limits.h>

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V];      // The output array. dist[i] will hold the shortest
                      // distance from src to i

    bool sptSet[V]; // sptSet[i] will true if vertex i is included in shortest
                    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
```

```

// Distance of source vertex from itself is always 0
dist[src] = 0;

// Find shortest path for all vertices
for (int count = 0; count < V-1; count++)
{
    // Pick the minimum distance vertex from the set of vertices not
    // yet processed. u is always equal to src in first iteration.
    int u = minDistance(dist, sptSet);

    // Mark the picked vertex as processed
    sptSet[u] = true;

    // Update dist value of the adjacent vertices of the picked vertex.
    for (int v = 0; v < V; v++)

        // Update dist[v] only if is not in sptSet, there is an edge from
        // u to v, and total weight of path from src to v through u is
        // smaller than current value of dist[v]
        if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
            && dist[u]+graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
}

// print the constructed distance array
printSolution(dist, V);
}

// driver program to test above function
int main()
{
    /* Let us create the example graph discussed above */
    int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                       {4, 0, 8, 0, 0, 0, 0, 11, 0},
                       {0, 8, 0, 7, 0, 4, 0, 0, 2},
                       {0, 0, 7, 0, 9, 14, 0, 0, 0},
                       {0, 0, 0, 9, 0, 10, 0, 0, 0},
                       {0, 0, 4, 14, 10, 0, 2, 0, 0},
                       {0, 0, 0, 0, 2, 0, 1, 6},
                       {8, 11, 0, 0, 0, 0, 1, 0, 7},
                       {0, 0, 2, 0, 0, 0, 6, 7, 0}
                      };

    dijkstra(graph, 0);

    return 0;
}

```

[Run on IDE](#)

Java

```

// A Java program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph
import java.util.*;
import java.lang.*;
import java.io.*;

class ShortestPath
{
    // A utility function to find the vertex with minimum distance value,
    // from the set of vertices not yet included in shortest path tree
    static final int V=9;
    int minDistance(int dist[], Boolean sptSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index=-1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min)
                {

```

```

        min = dist[v];
        min_index = v;
    }

    return min_index;
}

// A utility function to print the constructed distance array
void printSolution(int dist[], int n)
{
    System.out.println("Vertex   Distance from Source");
    for (int i = 0; i < V; i++)
        System.out.println(i+" \t\t "+dist[i]);
}

// Function that implements Dijkstra's single source shortest path
// algorithm for a graph represented using adjacency matrix
// representation
void dijkstra(int graph[][], int src)
{
    int dist[] = new int[V]; // The output array. dist[i] will hold
                           // the shortest distance from src to i

    // sptSet[i] will true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized
    Boolean sptSet[] = new Boolean[V];

    // Initialize all distances as INFINITE and sptSet[] as false
    for (int i = 0; i < V; i++)
    {
        dist[i] = Integer.MAX_VALUE;
        sptSet[i] = false;
    }

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V-1; count++)
    {
        // Pick the minimum distance vertex from the set of vertices
        // not yet processed. u is always equal to src in first
        // iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the
        // picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an
            // edge from u to v, and total weight of path from src to
            // v through u is smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v]!=0 &&
                dist[u] != Integer.MAX_VALUE &&
                dist[u]+graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist, V);
}

// Driver method
public static void main (String[] args)
{
    /* Let us create the example graph discussed above */
    int graph[][] = new int[][]{{0, 4, 0, 0, 0, 0, 0, 8, 0},
                               {4, 0, 8, 0, 0, 0, 0, 11, 0},
                               {0, 8, 0, 7, 0, 4, 0, 0, 2},
                               {0, 0, 7, 0, 9, 14, 0, 0, 0},
                               {0, 0, 0, 9, 0, 10, 0, 0, 0},
}

```

```

{0, 0, 4, 14, 10, 0, 2, 0, 0},
{0, 0, 0, 0, 0, 2, 0, 1, 6},
{8, 11, 0, 0, 0, 0, 1, 0, 7},
{0, 0, 2, 0, 0, 0, 6, 7, 0}
};

ShortestPath t = new ShortestPath();
t.dijkstra(graph, 0);
}

//This code is contributed by Aakash Hasija

```

[Run on IDE](#)

Output:

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

Notes:

- 1) The code calculates shortest distance, but doesn't calculate the path information. We can create a parent array, update the parent array when distance is updated (like [prim's implementation](#)) and use it show the shortest path from source to different vertices.
- 2) The code is for undirected graph, same dijkstra function can be used for directed graphs also.
- 3) The code finds shortest distances from source to all vertices. If we are interested only in shortest distance from source to a single target, we can break the for loop when the picked minimum distance vertex is equal to target (Step 3.a of algorithm).
- 4) Time Complexity of the implementation is $O(V^2)$. If the input graph is represented using [adjacency list](#), it can be reduced to $O(E \log V)$ with the help of binary heap. Please see [Dijkstra's Algorithm for Adjacency List Representation](#) for more details.
- 5) Dijkstra's algorithm doesn't work for graphs with negative weight edges. For graphs with negative weight edges, [Bellman–Ford algorithm](#) can be used, we will soon be discussing it as a separate post.

[Dijkstra's Algorithm for Adjacency List Representation](#)

[Printing Paths in Dijkstra's Shortest Path Algorithm](#)

[Dijkstra's shortest path algorithm using set in STL](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Graph](#) [Greedy](#) [Dijkstra](#) [Graph](#) [Greedy Algorithm](#) [shortest path](#)

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 57 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Greedy Algorithms | Set 8 (Dijkstra's Algorithm for Adjacency List Representation)

We recommend to read following two posts as a prerequisite of this post.

1. [Greedy Algorithms | Set 7 \(Dijkstra's shortest path algorithm\)](#)
2. [Graph and its representations](#)

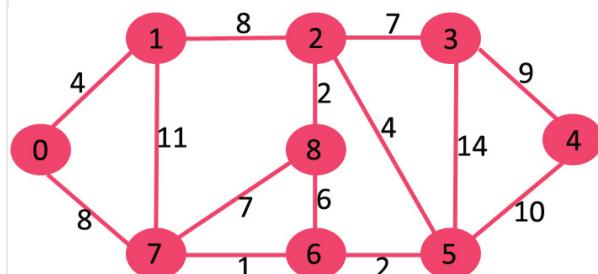
We have discussed Dijkstra's algorithm and its implementation for adjacency matrix representation of graphs. The time complexity for the matrix representation is $O(V^2)$. In this post, $O(E\log V)$ algorithm for adjacency list representation is discussed.

As discussed in the previous post, in Dijkstra's algorithm, two sets are maintained, one set contains list of vertices already included in SPT (Shortest Path Tree), other set contains vertices not yet included. With adjacency list representation, all vertices of a graph can be traversed in $O(V+E)$ time using BFS. The idea is to traverse all vertices of graph using BFS and use a Min Heap to store the vertices not yet included in SPT (or the vertices for which shortest distance is not finalized yet). Min Heap is used as a priority queue to get the minimum distance vertex from set of not yet included vertices. Time complexity of operations like extract-min and decrease-key value is $O(\log V)$ for Min Heap.

Following are the detailed steps.

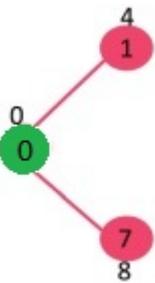
- 1) Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and distance value of the vertex.
- 2) Initialize Min Heap with source vertex as root (the distance value assigned to source vertex is 0). The distance value assigned to all other vertices is INF (infinite).
- 3) While Min Heap is not empty, do following
 - a) Extract the vertex with minimum distance value node from Min Heap. Let the extracted vertex be u .
 - b) For every adjacent vertex v of u , check if v is in Min Heap. If v is in Min Heap and distance value is more than weight of $u-v$ plus distance value of u , then update the distance value of v .

Let us understand with the following example. Let the given source vertex be 0

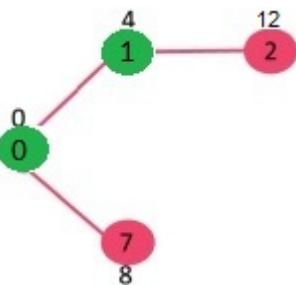


Initially, distance value of source vertex is 0 and INF (infinite) for all other vertices. So source vertex is extracted from Min Heap and distance values of vertices adjacent to 0 (1 and 7) are updated. Min Heap contains all vertices except vertex 0.

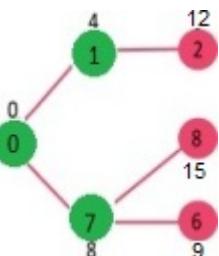
The vertices in green color are the vertices for which minimum distances are finalized and are not in Min Heap



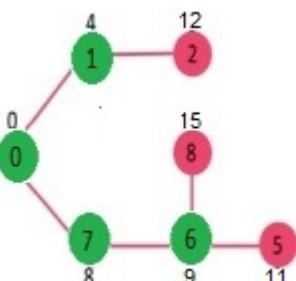
Since distance value of vertex 1 is minimum among all nodes in Min Heap, it is extracted from Min Heap and distance values of vertices adjacent to 1 are updated (distance is updated if the a vertex is not in Min Heap and distance through 1 is shorter than the previous distance). Min Heap contains all vertices except vertex 0 and 1.



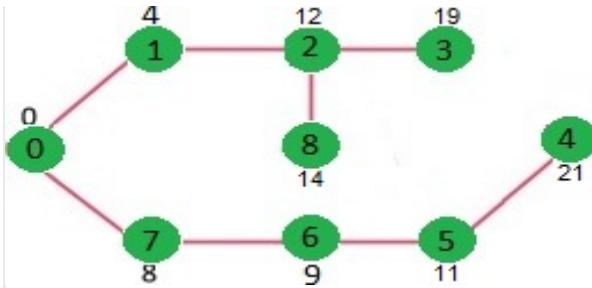
Pick the vertex with minimum distance value from min heap. Vertex 7 is picked. So min heap now contains all vertices except 0, 1 and 7. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance from min heap. Vertex 6 is picked. So min heap now contains all vertices except 0, 1, 7 and 6. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



Above steps are repeated till min heap doesn't become empty. Finally, we get the following shortest path tree.



```
// C / C++ program for Dijkstra's shortest path algorithm for adjacency
// list representation of graph

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// A structure to represent a node in adjacency list
struct AdjListNode
{
    int dest;
    int weight;
    struct AdjListNode* next;
};

// A structure to represent an adjacency list
struct AdjList
{
    struct AdjListNode *head; // pointer to head node of list
};

// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph
{
    int V;
    struct AdjList* array;
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest, int weight)
{
    struct AdjListNode* newNode =
        (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = NULL;
    return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;

    // Create an array of adjacency lists. Size of array will be V
    graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));

    // Initialize each adjacency list as empty by making head as NULL
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest, int weight)
{
    // Add an edge from src to dest. A new node is added to the adjacency
    // list of src. The node is added at the begining
    struct AdjListNode* newNode = newAdjListNode(dest, weight);
}
```

```

newNode->next = graph->array[src].head;
graph->array[src].head = newNode;

// Since graph is undirected, add an edge from dest to src also
newNode = newAdjListNode(src, weight);
newNode->next = graph->array[dest].head;
graph->array[dest].head = newNode;
}

// Structure to represent a min heap node
struct MinHeapNode
{
    int v;
    int dist;
};

// Structure to represent a min heap
struct MinHeap
{
    int size;      // Number of heap nodes present currently
    int capacity; // Capacity of min heap
    int *pos;      // This is needed for decreaseKey()
    struct MinHeapNode **array;
};

// A utility function to create a new Min Heap Node
struct MinHeapNode* newMinHeapNode(int v, int dist)
{
    struct MinHeapNode* minHeapNode =
        (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
    minHeapNode->v = v;
    minHeapNode->dist = dist;
    return minHeapNode;
}

// A utility function to create a Min Heap
struct MinHeap* createMinHeap(int capacity)
{
    struct MinHeap* minHeap =
        (struct MinHeap*) malloc(sizeof(struct MinHeap));
    minHeap->pos = (int*) malloc(capacity * sizeof(int));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array =
        (struct MinHeapNode**) malloc(capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}

// A utility function to swap two nodes of min heap. Needed for min heapify
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// A standard function to heapify at given idx
// This function also updates position of nodes when they are swapped.
// Position is needed for decreaseKey()
void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest, left, right;
    smallest = idx;
    left = 2 * idx + 1;
    right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->array[left]->dist < minHeap->array[smallest]->dist )
        smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->dist < minHeap->array[smallest]->dist )
        smallest = right;
}

```

```

if (smallest != idx)
{
    // The nodes to be swapped in min heap
    MinHeapNode *smallestNode = minHeap->array[smallest];
    MinHeapNode *idxNode = minHeap->array[idx];

    // Swap positions
    minHeap->pos[smallestNode->v] = idx;
    minHeap->pos[idxNode->v] = smallest;

    // Swap nodes
    swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);

    minHeapify(minHeap, smallest);
}

// A utility function to check if the given minHeap is ampty or not
int isEmpty(struct MinHeap* minHeap)
{
    return minHeap->size == 0;
}

// Standard function to extract minimum node from heap
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    if (isEmpty(minHeap))
        return NULL;

    // Store the root node
    struct MinHeapNode* root = minHeap->array[0];

    // Replace root node with last node
    struct MinHeapNode* lastNode = minHeap->array[minHeap->size - 1];
    minHeap->array[0] = lastNode;

    // Update position of last node
    minHeap->pos[root->v] = minHeap->size-1;
    minHeap->pos[lastNode->v] = 0;

    // Reduce heap size and heapify root
    --minHeap->size;
    minHeapify(minHeap, 0);

    return root;
}

// Function to decreasy dist value of a given vertex v. This function
// uses pos[] of min heap to get the current index of node in min heap
void decreaseKey(struct MinHeap* minHeap, int v, int dist)
{
    // Get the index of v in heap array
    int i = minHeap->pos[v];

    // Get the node and update its dist value
    minHeap->array[i]->dist = dist;

    // Travel up while the complete tree is not hepified.
    // This is a O(Logn) loop
    while (i && minHeap->array[i]->dist < minHeap->array[(i - 1) / 2]->dist)
    {
        // Swap this node with its parent
        minHeap->pos[minHeap->array[i]->v] = (i-1)/2;
        minHeap->pos[minHeap->array[(i-1)/2]->v] = i;
        swapMinHeapNode(&minHeap->array[i], &minHeap->array[(i - 1) / 2]);

        // move to parent index
        i = (i - 1) / 2;
    }

    // A utility function to check if a given vertex
    // 'v' is in min heap or not
    bool isInMinHeap(struct MinHeap *minHeap, int v)
}

```

```

{
    if (minHeap->pos[v] < minHeap->size)
        return true;
    return false;
}

// A utility function used to print the solution
void printArr(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

// The main function that calculates distances of shortest paths from src to all
// vertices. It is a O(ELogV) function
void dijkstra(struct Graph* graph, int src)
{
    int V = graph->V; // Get the number of vertices in graph
    int dist[V]; // dist values used to pick minimum weight edge in cut

    // minHeap represents set E
    struct MinHeap* minHeap = createMinHeap(V);

    // Initialize min heap with all vertices. dist value of all vertices
    for (int v = 0; v < V; ++v)
    {
        dist[v] = INT_MAX;
        minHeap->array[v] = newMinHeapNode(v, dist[v]);
        minHeap->pos[v] = v;
    }

    // Make dist value of src vertex as 0 so that it is extracted first
    minHeap->array[src] = newMinHeapNode(src, dist[src]);
    minHeap->pos[src] = src;
    dist[src] = 0;
    decreaseKey(minHeap, src, dist[src]);

    // Initially size of min heap is equal to V
    minHeap->size = V;

    // In the followin loop, min heap contains all nodes
    // whose shortest distance is not yet finalized.
    while (!isEmpty(minHeap))
    {
        // Extract the vertex with minimum distance value
        struct MinHeapNode* minHeapNode = extractMin(minHeap);
        int u = minHeapNode->v; // Store the extracted vertex number

        // Traverse through all adjacent vertices of u (the extracted
        // vertex) and update their distance values
        struct AdjListNode* pCrawl = graph->array[u].head;
        while (pCrawl != NULL)
        {
            int v = pCrawl->dest;

            // If shortest distance to v is not finalized yet, and distance to v
            // through u is less than its previously calculated distance
            if (isInMinHeap(minHeap, v) && dist[u] != INT_MAX &&
                pCrawl->weight + dist[u] < dist[v])
            {
                dist[v] = dist[u] + pCrawl->weight;

                // update distance value in min heap also
                decreaseKey(minHeap, v, dist[v]);
            }
            pCrawl = pCrawl->next;
        }
    }

    // print the calculated shortest distances
    printArr(dist, V);
}

```

```

// Driver program to test above functions
int main()
{
    // create the graph given in above figure
    int V = 9;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 4);
    addEdge(graph, 0, 7, 8);
    addEdge(graph, 1, 2, 8);
    addEdge(graph, 1, 7, 11);
    addEdge(graph, 2, 3, 7);
    addEdge(graph, 2, 8, 2);
    addEdge(graph, 2, 5, 4);
    addEdge(graph, 3, 4, 9);
    addEdge(graph, 3, 5, 14);
    addEdge(graph, 4, 5, 10);
    addEdge(graph, 5, 6, 2);
    addEdge(graph, 6, 7, 1);
    addEdge(graph, 6, 8, 6);
    addEdge(graph, 7, 8, 7);

    dijkstra(graph, 0);

    return 0;
}

```

Run on IDE

Output:

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

Time Complexity: The time complexity of the above code/algorithm looks $O(V^2)$ as there are two nested while loops. If we take a closer look, we can observe that the statements in inner loop are executed $O(V+E)$ times (similar to BFS). The inner loop has decreaseKey() operation which takes $O(\log V)$ time. So overall time complexity is $O(E+V)*O(\log V)$ which is $O((E+V)*\log V) = O(E\log V)$

Note that the above code uses Binary Heap for Priority Queue implementation. Time complexity can be reduced to $O(E + V\log V)$ using Fibonacci Heap. The reason is, Fibonacci Heap takes $O(1)$ time for decrease-key operation while Binary Heap takes $O(\log n)$ time.

Notes:

- 1) The code calculates shortest distance, but doesn't calculate the path information. We can create a parent array, update the parent array when distance is updated (like [prim's implementation](#)) and use it show the shortest path from source to different vertices.
- 2) The code is for undirected graph, same dijkstra function can be used for directed graphs also.
- 3) The code finds shortest distances from source to all vertices. If we are interested only in shortest distance from source to a single target, we can break the for loop when the picked minimum distance vertex is equal to target (Step 3.a of algorithm).

4) Dijkstra's algorithm doesn't work for graphs with negative weight edges. For graphs with negative weight edges, [Bellman–Ford algorithm](#) can be used, we will soon be discussing it as a separate post.

Printing Paths in Dijkstra's Shortest Path Algorithm

Dijkstra's shortest path algorithm using set in STL

References:

Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Algorithms by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Graph](#) [Greedy](#) [Dijkstra](#) [shortest path](#)

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

3.9

Average Difficulty : 3.9/5.0
Based on 17 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Job Sequencing Problem | Set 1 (Greedy Algorithm)

Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

Examples:

Input: Four Jobs with following deadlines and profits

JobID	Deadline	Profit
a	4	20
b	1	10
c	1	40
d	1	30

Output: Following is maximum profit sequence of jobs

c, a

Input: Five Jobs with following deadlines and profits

JobID	Deadline	Profit
a	2	100
b	1	19
c	2	27
d	1	25
e	3	15

Output: Following is maximum profit sequence of jobs

c, a, e

We strongly recommend to minimize your browser and try this yourself first.

A **Simple Solution** is to generate all subsets of given set of jobs and check individual subset for feasibility of jobs in that subset. Keep track of maximum profit among all feasible subsets. The time complexity of this solution is exponential.

This is a standard [Greedy Algorithm](#) problem. Following is algorithm.

- 1) Sort all jobs in decreasing order of profit.
- 2) Initialize the result sequence as first job in sorted jobs.
- 3) Do following for remaining n-1 jobs
 -a) If the current job can fit in the current result sequence without missing the deadline, add current job to the result.
 - Else ignore the current job.

The Following is C++ implementation of above algorithm.

```
// Program to find the maximum profit job sequence from a given array
// of jobs with deadlines and profits
#include<iostream>
#include<algorithm>
using namespace std;

// A structure to represent a job
struct Job
{
    char id;      // Job Id
    int dead;     // Deadline of job
    int profit;   // Profit if job is over before or on deadline
};

// This function is used for sorting all jobs according to profit
bool comparison(Job a, Job b)
{
    return (a.profit > b.profit);
}

// Returns minimum number of platforms required
void printJobScheduling(Job arr[], int n)
{
    // Sort all jobs according to decreasing order of profit
    sort(arr, arr+n, comparison);

    int result[n]; // To store result (Sequence of jobs)
    bool slot[n]; // To keep track of free time slots

    // Initialize all slots to be free
    for (int i=0; i<n; i++)
        slot[i] = false;

    // Iterate through all given jobs
    for (int i=0; i<n; i++)
    {
        // Find a free slot for this job (Note that we start
        // from the last possible slot)
        for (int j=min(n, arr[i].dead)-1; j>=0; j--)
        {
            // Free slot found
            if (slot[j]==false)
            {
                result[j] = i; // Add this job to result
                slot[j] = true; // Make this slot occupied
                break;
            }
        }
    }

    // Print the result
    for (int i=0; i<n; i++)
        if (slot[i])
            cout << arr[result[i]].id << " ";
}

// Driver program to test methods
int main()
{
    Job arr[] = { {'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27},
                 {'d', 1, 25}, {'e', 3, 15} };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Following is maximum profit sequence of jobs\n";
    printJobScheduling(arr, n);
    return 0;
}
```

Run on IDE

Output:

Following is maximum profit sequence of jobs
c a e

Time Complexity of the above solution is $O(n^2)$. It can be optimized using Disjoint Set Data Structure. Please refer below post for details.

[Job Sequencing Problem | Set 2 \(Using Disjoint Set\)](#)

Sources:

http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-204-computer-algorithms-in-systems-engineering-spring-2010/lecture-notes/MIT1_204S10_lec10.pdf

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



GATE CS Corner Company Wise Coding Practice

[Greedy](#) [Greedy Algorithm](#)

Related Posts:

- Minimize the sum of product of two arrays with permutations allowed
- Find maximum sum possible equal sum of three stacks
- Minimum sum of two numbers formed from digits of an array
- Maximize array sum after K negations | Set 2
- Minimum edges to reverse to make path from a source to a destination
- Minimum Cost to cut a board into squares
- Maximize array sum after K negations | Set 1
- Job Sequencing Problem | Set 2 (Using Disjoint Set)

[\(Login to Rate and Mark\)](#)**2.7**Average Difficulty : **2.7/5.0**
Based on **21** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Greedy Algorithms

Question 1

Which of the following standard algorithms is not a Greedy algorithm?

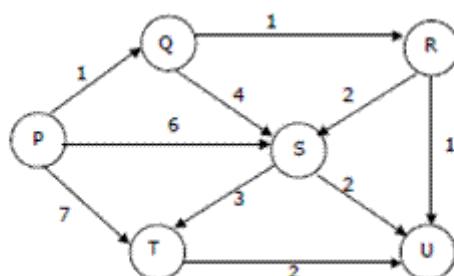
- A Dijkstra's shortest path algorithm
- B Prim's algorithm
- C Kruskal algorithm
- D Huffman Coding
- E Bellmen Ford Shortest path algorithm

Greedy Algorithms

[Discuss it](#)

Question 2

Suppose we run Dijkstra's single source shortest-path algorithm on the following edge weighted directed graph with vertex P as the source. In what order do the nodes get included into the set of vertices for which the shortest path distances are finalized? (GATE CS 2004)



- A P, Q, R, S, T, U

B

P, Q, R, U, S, T

C

P, Q, R, U, T, S

D

P, Q, T, R, U, S

Greedy Algorithms**Discuss it****Question 3**

A networking company uses a compression technique to encode the message before transmitting over the network. Suppose the message contains the following characters with their frequency:

character	Frequency
-----------	-----------

a	5
b	9
c	12
d	13
e	16
f	45

[Run on IDE](#)

Note : Each character in input message takes 1 byte. If the compression technique used is Huffman Coding, how many bits will be saved in the message?

A

224

B

800

C

576

D

324

Greedy Algorithms**Discuss it****Question 4**

What is the time complexity of Huffman Coding?

A $O(N)$ **B** $O(N \log N)$

C $O(N(\log N)^2)$ **D** $O(N^2)$ **Greedy Algorithms****Discuss it****Question 5**

In question #2, which of the following represents the word "dead"?

A

1011111100101

B

0100000011010

C

Both A and B

D

None of these

Greedy Algorithms**Discuss it****Question 6**

Which of the following is true about Kruskal and Prim MST algorithms? Assume that Prim is implemented for adjacency list representation using Binary Heap and Kruskal is implemented using union by rank.

A

Worst case time complexity of both algorithms is same.

B

Worst case time complexity of Kruskal is better than Prim

C

Worst case time complexity of Prim is better than Kruskal

Greedy Algorithms**Discuss it****Question 7**

Which of the following is true about Huffman Coding.

- A
- B
- C
- D

Huffman coding may become lossy in some cases
Huffman Codes may not be optimal lossless codes in some cases
In Huffman coding, no code is prefix of any other code.
All of the above

Greedy Algorithms

Discuss it

Question 8

Suppose the letters a, b, c, d, e, f have probabilities $1/2$, $1/4$, $1/8$, $1/16$, $1/32$, $1/32$ respectively.
Which of the following is the Huffman code for the letter a, b, c, d, e, f?

- A 0, 10, 110, 1110, 11110, 11111
- B 11, 10, 011, 010, 001, 000
- C 11, 10, 01, 001, 0001, 0000
- D 110, 100, 010, 000, 001, 111

Greedy Algorithms GATE-CS-2007

Discuss it

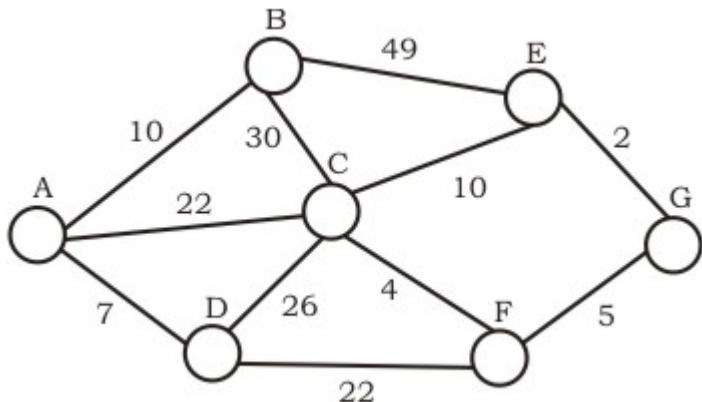
Question 9

Suppose the letters a, b, c, d, e, f have probabilities $1/2$, $1/4$, $1/8$, $1/16$, $1/32$, $1/32$ respectively. What is the average length of Huffman codes?

- A 3
- B 2.1875
- C 2.25
- D 1.9375

Greedy Algorithms GATE-CS-2007**Discuss it****Question 10**

Consider the undirected graph below:



Using Prim's algorithm to

construct a minimum spanning tree starting with node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree?

A

(E, G), (C, F), (F, G), (A, D), (A, B), (A, C)

B

(A, D), (A, B), (A, C), (C, F), (G, E), (F, G)

C

(A, B), (A, D), (D, F), (F, G), (G, E), (F, C)

D

(A, D), (A, B), (D, F), (F, C), (F, G), (G, E)

Greedy Algorithms GATE-IT-2004**Discuss it**

There are 10 questions to complete.

GATE CS Corner



Download

Free Download

unzipper.com



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Greedy Algorithm to find Minimum number of Coins

Given a value V, if we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?

Examples:

```
Input: V = 70
Output: 2
We need a 50 Rs note and a 20 Rs note.
```

```
Input: V = 121
Output: 3
We need a 100 Rs note, a 20 Rs note and a
1 Rs coin.
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is simple Greedy Algorithm. Start from largest possible denomination and keep adding denominations while remaining value is greater than 0. Below is complete algorithm.

- 1) Initialize result as empty.
- 2) find the largest denomination that is smaller than V.
- 3) Add found denomination to result. Subtract value of found denomination from V.
- 4) If V becomes 0, then print result.
Else repeat steps 2 and 3 for new value of V

Below is C++ implementation of above algorithm.

```
// C++ program to find minimum number of denominations
#include <bits/stdc++.h>
using namespace std;

// All denominations of Indian Currency
int deno[] = {1, 2, 5, 10, 20, 50, 100, 500, 1000};
int n = sizeof(deno)/sizeof(deno[0]);

// Driver program
void findMin(int V)
{
    // Initialize result
    vector<int> ans;

    // Traverse through all denomination
    for (int i=n-1; i>=0; i--)
    {
        // Find denominations
        while (V >= deno[i])
        {
            V -= deno[i];
            ans.push_back(deno[i]);
        }
    }
}
```

```

    }

    // Print result
    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << " ";
}

// Driver program
int main()
{
    int n = 93;
    cout << "Following is minimal number of change for " << n << " is ";
    findMin(n);
    return 0;
}

```

[Run on IDE](#)

Output:

```
Following is minimal number of change for 93 is 50 20 20 2 1
```

Note that above approach may not work for all denominations. For example, it doesn't work for denominations {9, 6, 5, 1} and V = 11. The above approach would print 9, 1 and 1. But we can use 2 denominations 5 and 6.

For general input, we use below dynamic programming approach.

[Find minimum number of coins that make a given value](#)

Thanks to Utkarsh for providing above solution here.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

(Login to Rate and Mark)

Average Rating : **1.5/5.0**

Average Difficulty : **1.2/5.0**

1.2

Based on **14** vote(s)



Add to TODO List

Mark as DONE

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

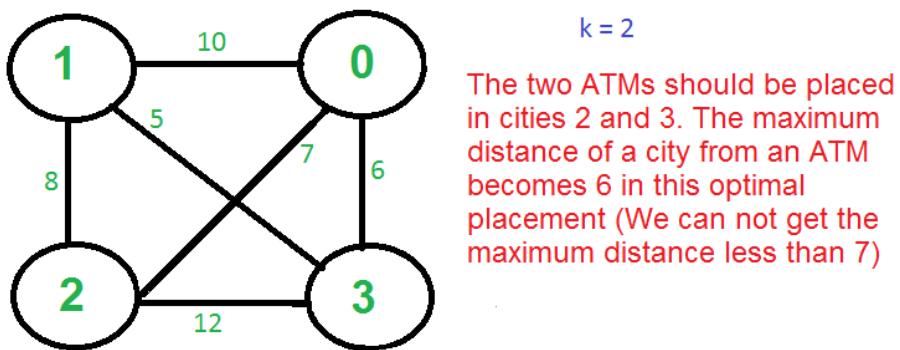


[Login/Register](#)

K Centers Problem | Set 1 (Greedy Approximate Algorithm)

Given n cities and distances between every pair of cities, select k cities to place warehouses (or ATMs or Cloud Server) such that the maximum distance of a city to a warehouse (or ATM or Cloud Server) is minimized.

For example consider the following four cities, 0, 1, 2 and 3 and distances between them, how do place 2 ATMs among these 4 cities so that the maximum distance of a city to an ATM is minimized.



There is no polynomial time solution available for this problem as the problem is a known NP-Hard problem. There is a polynomial time Greedy approximate algorithm, the greedy algorithm provides a solution which is never worse than twice the optimal solution. The greedy solution works only if the distances between cities follow [Triangular Inequality](#) (Distance between two points is always smaller than sum of distances through a third point).

The 2-Approximate Greedy Algorithm:

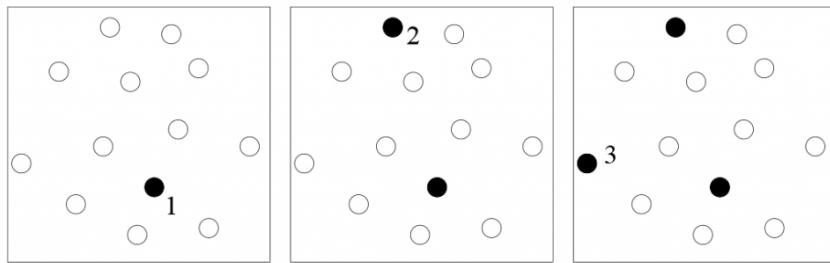
1) Choose the first center arbitrarily.

2) Choose remaining $k-1$ centers using the following criteria.

Let $c_1, c_2, c_3, \dots, c_i$ be the already chosen centers. Choose $(i+1)$ 'th center by picking the city which is farthest from already selected centers, i.e, the point p which has following value as maximum

$$\text{Min}[\text{dist}(p, c_1), \text{dist}(p, c_2), \text{dist}(p, c_3), \dots, \text{dist}(p, c_i)]$$

The following diagram taken from [here](#) illustrates above algorithm.



Example ($k = 3$ in the above shown Graph)

- Let the first arbitrarily picked vertex be 0.
- The next vertex is 1 because 1 is the farthest vertex from 0.
- Remaining cities are 2 and 3. Calculate their distances from already selected centers (0 and 1). The greedy algorithm basically calculates following values.

Minimum of all distances from 2 to already considered centers

$$\text{Min}[\text{dist}(2, 0), \text{dist}(2, 1)] = \text{Min}[7, 8] = 7$$

Minimum of all distances from 3 to already considered centers

$$\text{Min}[\text{dist}(3, 0), \text{dist}(3, 1)] = \text{Min}[6, 5] = 5$$

After computing the above values, the city 2 is picked as the value corresponding to 2 is maximum.

Note that the greedy algorithm doesn't give best solution for $k = 2$ as this is just an approximate algorithm with bound as twice of optimal.

Proof that the above greedy algorithm is 2 approximate.

Let OPT be the maximum distance of a city from a center in the Optimal solution. We need to show that the maximum distance obtained from Greedy algorithm is $2 \cdot \text{OPT}$.

The proof can be done using contradiction.

- Assume that the distance from the furthest point to all centers is $> 2 \cdot \text{OPT}$.
- This means that distances between all centers are also $> 2 \cdot \text{OPT}$.
- We have $k + 1$ points with distances $> 2 \cdot \text{OPT}$ between every pair.
- Each point has a center of the optimal solution with distance $\leq \text{OPT}$ to it.
- There exists a pair of points with the same center X in the optimal solution (pigeonhole principle: k optimal centers, $k+1$ points)
- The distance between them is at most $2 \cdot \text{OPT}$ (triangle inequality) which is a contradiction.

Source:

<http://algo2.iti.kit.edu/vanstee/courses/kcenter.pdf>

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



GATE CS Corner Company Wise Coding Practice

[Graph](#) [Greedy](#) [Greedy Algorithm](#)

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

4.3

Average Difficulty : 4.3/5.0
Based on 11 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

HANDBOOK OF ALGORITHMS

Section
Divide & Conquer

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Divide and Conquer | Set 1 (Introduction)

Like [Greedy](#) and [Dynamic Programming](#), Divide and Conquer is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.

1. **Divide:** Break the given problem into subproblems of same type.
2. **Conquer:** Recursively solve these subproblems
3. **Combine:** Appropriately combine the answers

Following are some standard algorithms that are Divide and Conquer algorithms.

- 1) **Binary Search** is a searching algorithm. In each step, the algorithm compares the input element x with the value of the middle element in array. If the values match, return the index of middle. Otherwise, if x is less than the middle element, then the algorithm recurs for left side of middle element, else recurs for right side of middle element.
- 2) **Quicksort** is a sorting algorithm. The algorithm picks a pivot element, rearranges the array elements in such a way that all elements smaller than the picked pivot element move to left side of pivot, and all greater elements move to right side. Finally, the algorithm recursively sorts the subarrays on left and right of pivot element.
- 3) **Merge Sort** is also a sorting algorithm. The algorithm divides the array in two halves, recursively sorts them and finally merges the two sorted halves.
- 4) **Closest Pair of Points** The problem is to find the closest pair of points in a set of points in x-y plane. The problem can be solved in $O(n^2)$ time by calculating distances of every pair of points and comparing the distances to find the minimum. The Divide and Conquer algorithm solves the problem in $O(n \log n)$ time.
- 5) **Strassen's Algorithm** is an efficient algorithm to multiply two matrices. A simple method to multiply two matrices need 3 nested loops and is $O(n^3)$. Strassen's algorithm multiplies two matrices in $O(n^{2.8974})$ time.
- 6) **Cooley–Tukey Fast Fourier Transform (FFT) algorithm** is the most common algorithm for FFT. It is a divide and conquer algorithm which works in $O(n \log n)$ time.
- 7) **Karatsuba algorithm for fast multiplication** it does multiplication of two n -digit numbers in at most $3n^{\log_2 3} \approx 3n^{1.585}$ single-digit multiplications in general (and exactly $n^{\log_2 3}$ when n is a power of 2). It is therefore faster than the [classical](#) algorithm, which requires n^2 single-digit products. If $n = 2^{10} = 1024$, in particular, the exact counts are $3^{10} = 59,049$ and $(2^{10})^2 = 1,048,576$, respectively.

We will publishing above algorithms in separate posts.

Divide and Conquer (D & C) vs Dynamic Programming (DP)

Both paradigms (D & C and DP) divide the given problem into subproblems and solve subproblems. How to choose

one of them for a given problem? Divide and Conquer should be used when same subproblems are not evaluated many times. Otherwise Dynamic Programming or Memoization should be used. For example, Binary Search is a Divide and Conquer algorithm, we never evaluate the same subproblems again. On the other hand, for calculating nth Fibonacci number, Dynamic Programming should be preferred (See [this](#) for details).

References

Algorithms by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani

Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

http://en.wikipedia.org/wiki/Karatsuba_algorithm

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Divide and Conquer](#) [Divide and Conquer](#)

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

2

Average Difficulty : 2/5.0
Based on 23 vote(s)



Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Write a program to calculate pow(x,n)

Below solution divides the problem into subproblems of size y/2 and call the subproblems recursively.

```
#include<stdio.h>

/* Function to calculate x raised to the power y */
int power(int x, unsigned int y)
{
    if( y == 0)
        return 1;
    else if (y%2 == 0)
        return power(x, y/2)*power(x, y/2);
    else
        return x*power(x, y/2)*power(x, y/2);

}

/* Program to test function power */
int main()
{
    int x = 2;
    unsigned int y = 3;

    printf("%d", power(x, y));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Time Complexity: O(n)

Space Complexity: O(1)

Algorithmic Paradigm: Divide and conquer.

Above function can be optimized to O(logn) by calculating power(x, y/2) only once and storing it.

```
/* Function to calculate x raised to the power y in O(logn)*/
int power(int x, unsigned int y)
{
    int temp;
    if( y == 0)
        return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else
        return x*temp*temp;
}
```

[Run on IDE](#)

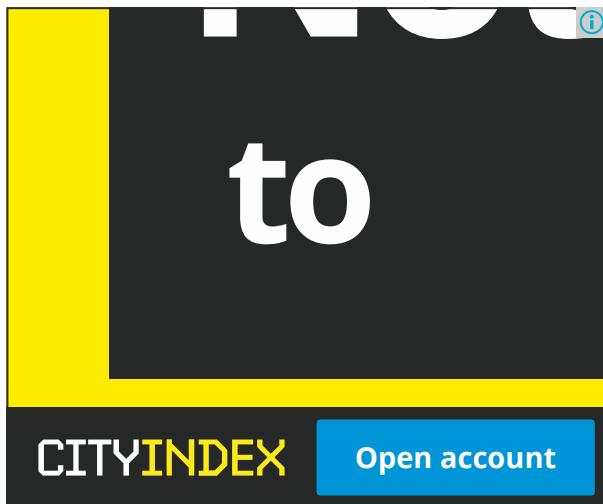
Time Complexity of optimized solution: O(logn)

Let us extend the pow function to work for negative y and float x.

```
/* Extended version of power function that can work
   for float x and negative y*/
#include<stdio.h>

float power(float x, int y)
{
    float temp;
    if( y == 0)
        return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else
    {
        if(y > 0)
            return x*temp*temp;
        else
            return (temp*temp)/x;
    }
}

/* Program to test function power */
int main()
{
    float x = 2;
    int y = -3;
    printf("%f", power(x, y));
    getchar();
    return 0;
}
```

[Run on IDE](#)


GATE CS Corner Company Wise Coding Practice

[Divide and Conquer](#)
[Divide and Conquer](#)

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix

- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

2.4

Average Difficulty : 2.4/5.0
Based on 71 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Median of two sorted arrays

Question: There are 2 sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained after merging the above 2 arrays(i.e. array of length 2n). The complexity should be O(log(n))

We strongly recommend that you click here and practice it, before moving on to the solution.

Median: In probability theory and statistics, a median is described as the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.

The median of a finite list of numbers can be found by arranging all the numbers from lowest value to highest value and picking the middle one.

For getting the median of input array { 12, 11, 15, 10, 20 }, first sort the array. We get { 10, 11, 12, 15, 20 } after sorting. Median is the middle element of the sorted array which is 12.

There are different conventions to take median of an array with even number of elements, one can take the mean of the two middle values, or first middle value, or second middle value.

Let us see different methods to get the median of two sorted arrays of size n each. Since size of the set for which we are looking for median is even (2n), we are taking average of middle two numbers in all below solutions.

Method 1 (Simply count while Merging)

Use merge procedure of merge sort. Keep track of count while comparing elements of two arrays. If count becomes n(For 2n elements), we have reached the median. Take the average of the elements at indexes n-1 and n in the merged array. See the below implementation.

Implementation:

```
// A Simple Merge based O(n) solution to find median of
// two sorted arrays
#include <stdio.h>

/* This function returns median of ar1[] and ar2[].
Assumptions in this function:
Both ar1[] and ar2[] are sorted arrays
Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    int i = 0; /* Current index of i/p array ar1[] */
    int j = 0; /* Current index of i/p array ar2[] */
```

```

int count;
int m1 = -1, m2 = -1;

/* Since there are 2n elements, median will be average
   of elements at index n-1 and n in the array obtained after
   merging ar1 and ar2 */
for (count = 0; count <= n; count++)
{
    /*Below is to handle case where all elements of ar1[] are
       smaller than smallest(or first) element of ar2[]*/
    if (i == n)
    {
        m1 = m2;
        m2 = ar2[0];
        break;
    }

    /*Below is to handle case where all elements of ar2[] are
       smaller than smallest(or first) element of ar1[]*/
    else if (j == n)
    {
        m1 = m2;
        m2 = ar1[0];
        break;
    }

    if (ar1[i] < ar2[j])
    {
        m1 = m2; /* Store the prev median */
        m2 = ar1[i];
        i++;
    }
    else
    {
        m1 = m2; /* Store the prev median */
        m2 = ar2[j];
        j++;
    }
}

return (m1 + m2)/2;
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 12, 15, 26, 38};
    int ar2[] = {2, 13, 17, 30, 45};

    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
        printf("Median is %d", getMedian(ar1, ar2, n1));
    else
        printf("Doesn't work for arrays of unequal size");
    getchar();
    return 0;
}

```

[Run on IDE](#)

Output

Median is 16

Time Complexity: O(n)

Method 2 (By comparing the medians of two arrays)

This method works by first getting medians of the two sorted arrays and then comparing them.

Let ar1 and ar2 be the input arrays.

Algorithm:

- 1) Calculate the medians m1 and m2 of the input arrays ar1[] and ar2[] respectively.
- 2) If m1 and m2 both are equal then we are done.
 return m1 (or m2)
- 3) If m1 is greater than m2, then median is present in one of the below two subarrays.
 - a) From first element of ar1 to m1 (ar1[0...|_n/2_|])
 - b) From m2 to last element of ar2 (ar2[|_n/2_|...n-1])
- 4) If m2 is greater than m1, then median is present in one of the below two subarrays.
 - a) From m1 to last element of ar1 (ar1[|_n/2_|...n-1])
 - b) From first element of ar2 to m2 (ar2[0...|_n/2_|])
- 5) Repeat the above process until size of both the subarrays becomes 2.
- 6) If size of the two arrays is 2 then use below formula to get the median.
 Median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2

Example:

```
ar1[] = {1, 12, 15, 26, 38}
ar2[] = {2, 13, 17, 30, 45}
```

For above two arrays m1 = 15 and m2 = 17

For the above ar1[] and ar2[], m1 is smaller than m2. So median is present in one of the following two subarrays.

```
[15, 26, 38] and [2, 13, 17]
```

Let us repeat the process for above two subarrays:

```
m1 = 26 m2 = 13.
```

m1 is greater than m2. So the subarrays become

```
[15, 26] and [13, 17]
```

```
Now size is 2, so median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
= (max(15, 13) + min(26, 17))/2
= (15 + 17)/2
= 16
```

Implementation:

```
// A divide and conquer based efficient solution to find median
// of two sorted arrays of same size.
#include<bits/stdc++.h>
using namespace std;

int median(int [], int); /* to get median of a sorted array */
```

```

/* This function returns median of ar1[] and ar2[].
Assumptions in this function:
Both ar1[] and ar2[] are sorted arrays
Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    /* return -1 for invalid input */
    if (n <= 0)
        return -1;
    if (n == 1)
        return (ar1[0] + ar2[0])/2;
    if (n == 2)
        return (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;

    int m1 = median(ar1, n); /* get the median of the first array */
    int m2 = median(ar2, n); /* get the median of the second array */

    /* If medians are equal then return either m1 or m2 */
    if (m1 == m2)
        return m1;

    /* if m1 < m2 then median must exist in ar1[m1....] and
       ar2[....m2] */
    if (m1 < m2)
    {
        if (n % 2 == 0)
            return getMedian(ar1 + n/2 - 1, ar2, n - n/2 +1);
        return getMedian(ar1 + n/2, ar2, n - n/2);
    }

    /* if m1 > m2 then median must exist in ar1[....m1] and
       ar2[m2...] */
    if (n % 2 == 0)
        return getMedian(ar2 + n/2 - 1, ar1, n - n/2 + 1);
    return getMedian(ar2 + n/2, ar1, n - n/2);
}

/* Function to get median of a sorted array */
int median(int arr[], int n)
{
    if (n%2 == 0)
        return (arr[n/2] + arr[n/2-1])/2;
    else
        return arr[n/2];
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 2, 3, 6};
    int ar2[] = {4, 6, 8, 10};
    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
        printf("Median is %d", getMedian(ar1, ar2, n1));
    else
        printf("Doesn't work for arrays of unequal size");
    return 0;
}

```

Run on IDE

Output :

Median is 5

Time Complexity: O(logn)

Algorithmic Paradigm: Divide and Conquer

Median of two sorted arrays of different sizes

References:

<http://en.wikipedia.org/wiki/Median>

<http://ocw.alfaisal.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/30C68118-E436-4FE3-8C79-6BAFBB07D935/0/ps9sol.pdf> ds3etph5wn

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Divide and Conquer Searching Divide and Conquer

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

3.6

Average Difficulty : 3.6/5.0
Based on 149 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count Inversions in an array | Set 1 (Using Merge Sort)

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.

Formally speaking, two elements $a[i]$ and $a[j]$ form an inversion if $a[i] > a[j]$ and $i < j$

Example:

The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

We strongly recommend that you click here and practice it, before moving on to the solution.

METHOD 1 (Simple)

For each element, count number of elements which are on right side of it and are smaller than it.

```
#include <bits/stdc++.h>
int getInvCount(int arr[], int n)
{
    int inv_count = 0;
    for (int i = 0; i < n - 1; i++)
        for (int j = i+1; j < n; j++)
            if (arr[i] > arr[j])
                inv_count++;

    return inv_count;
}

/* Driver program to test above functions */
int main(int argc, char** argv)
{
    int arr[] = {1, 20, 6, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf(" Number of inversions are %d \n", getInvCount(arr, n));
    return 0;
}
```

[Run on IDE](#)

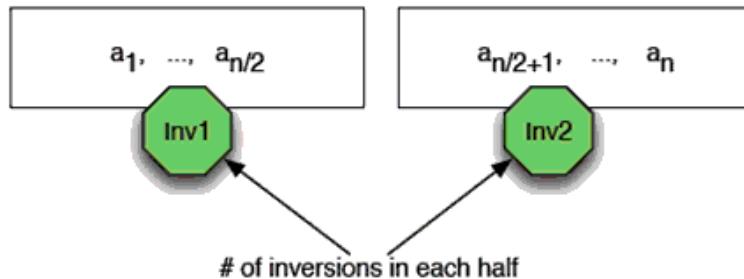
Output:

Number of inversions are 5

Time Complexity: $O(n^2)$

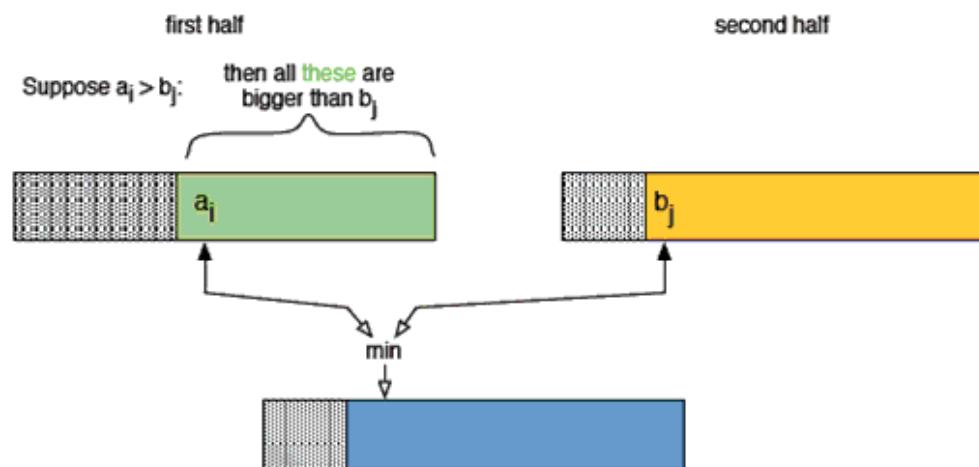
METHOD 2(Enhance Merge Sort)

Suppose we know the number of inversions in the left half and right half of the array (let be $inv1$ and $inv2$), what kinds of inversions are not accounted for in $Inv1 + Inv2$? The answer is – the inversions we have to count during the merge step. Therefore, to get number of inversions, we need to add number of inversions in left subarray, right subarray and merge().

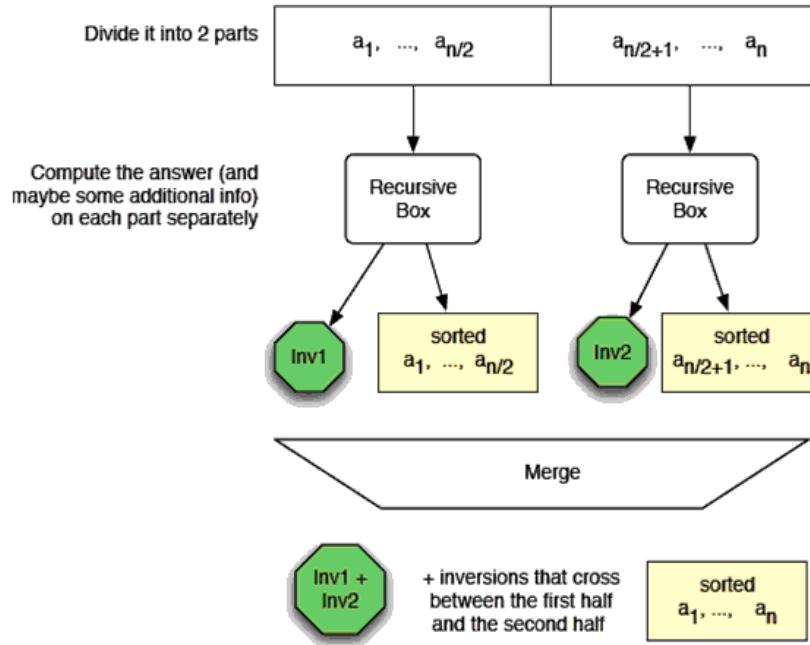


How to get number of inversions in merge()?

In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if $a[i]$ is greater than $a[j]$, then there are $(mid - i)$ inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray ($a[i+1], a[i+2] \dots a[mid]$) will be greater than $a[j]$



The complete picture:



Implementation:

```
#include <bits/stdc++.h>

int _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid, int right);

/* This function sorts the input array and returns the
   number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int *temp = (int *)malloc(sizeof(int)*array_size);
    return _mergeSort(arr, temp, 0, array_size - 1);
}

/* An auxiliary recursive function that sorts the input array and
   returns the number of inversions in the array. */
int _mergeSort(int arr[], int temp[], int left, int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and call _mergeSortAndCountInv()
           for each of the parts */
        mid = (right + left)/2;

        /* Inversion count will be sum of inversions in left-part, right-part
           and number of inversions in merging */
        inv_count = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }
    return inv_count;
}

/* This funt merges two sorted arrays and returns inversion count in
   the arrays.*/
int merge(int arr[], int temp[], int left, int mid, int right)
{
    int i, j, k;
    int inv_count = 0;

    i = left; /* i is index for left subarray*/
    j = mid; /* i is index for right subarray*/
    k = left;
```

```

k = left; /* i is index for resultant merged subarray*/
while ((i <= mid - 1) && (j <= right))
{
    if (arr[i] <= arr[j])
    {
        temp[k++] = arr[i++];
    }
    else
    {
        temp[k++] = arr[j++];
    }

    /*this is tricky -- see above explanation/diagram for merge()*/
    inv_count = inv_count + (mid - i);
}
}

/* Copy the remaining elements of left subarray
(if there are any) to temp*/
while (i <= mid - 1)
    temp[k++] = arr[i++];

/* Copy the remaining elements of right subarray
(if there are any) to temp*/
while (j <= right)
    temp[k++] = arr[j++];

/*Copy back the merged elements to original array*/
for (i=left; i <= right; i++)
    arr[i] = temp[i];

return inv_count;
}

/* Driver progra to test above functions */
int main(int argc, char** args)
{
    int arr[] = {1, 20, 6, 4, 5};
    printf(" Number of inversions are %d \n", mergeSort(arr, 5));
    getchar();
    return 0;
}

```

[Run on IDE](#)

Output:

Number of inversions are 5

Note that above code modifies (or sorts) the input array. If we want to count only inversions then we need to create a copy of original array and call mergeSort() on copy.

Time Complexity: O(nlogn)

Algorithmic Paradigm: Divide and Conquer

Asked in: [Adobe](#), [Amazon](#), [BankBazaar](#), [Flipkart](#), [Microsoft](#), [Myntra](#)

You may like to see.

[Count inversions in an array | Set 2 \(Using Self-Balancing BST\)](#)

[Counting Inversions using Set in C++ STL](#)

[Count inversions in an array | Set 3 \(Using BIT\)](#)

References:

<http://www.cs.umd.edu/class/fall2009/cmsc451/lectures/Lec08-inversions.pdf>

<http://www.cp.eng.chula.ac.th/~piak/teaching/algo/algo2008/count-inv.htm>

Please write comments if you find any bug in the above program/algorithm or other ways to solve the same problem.

Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Divide and Conquer Sorting Divide and Conquer inversion Merge Sort

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

3.6 Average Difficulty : **3.6/5.0**
Based on **144** vote(s)

Add to TODO List
 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Divide and Conquer | Set 2 (Closest Pair of Points)

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q .

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

The Brute force solution is $O(n^2)$, compute the distance between each pair and return the smallest. We can calculate the smallest distance in $O(n \log n)$ time using Divide and Conquer strategy. In this post, a $O(n \times (\log n)^2)$ approach is discussed. We will be discussing a $O(n \log n)$ approach in a separate post.

Algorithm

Following are the detailed steps of a $O(n (\log n)^2)$ algorithm.

Input: An array of n points $P[]$

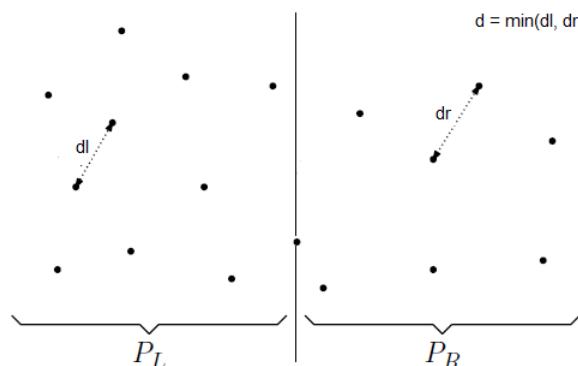
Output: The smallest distance between two points in the given array.

As a pre-processing step, input array is sorted according to x coordinates.

1) Find the middle point in the sorted array, we can take $P[n/2]$ as middle point.

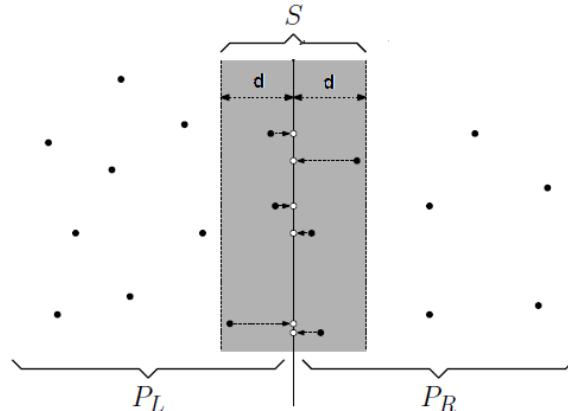
2) Divide the given array in two halves. The first subarray contains points from $P[0]$ to $P[n/2]$. The second subarray contains points from $P[n/2+1]$ to $P[n-1]$.

3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr . Find the minimum of dl and dr . Let the minimum be d .



4) From above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through

through $P[n/2]$ and find all points whose x coordinate is closer than d to the middle vertical line. Build an array $\text{strip}[]$ of all such points.



5) Sort the array $\text{strip}[]$ according to y coordinates. This step is $O(n \log n)$. It can be optimized to $O(n)$ by recursively sorting and merging.

6) Find the smallest distance in $\text{strip}[]$. This is tricky. From first look, it seems to be a $O(n^2)$ step, but it is actually $O(n)$. It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). See [this](#) for more analysis.

7) Finally return the minimum of d and distance calculated in above step (step 6)

Implementation

Following is C/C++ implementation of the above algorithm.

```
// A divide and conquer program in C/C++ to find the smallest distance from a
// given set of points.

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>

// A structure to represent a Point in 2D plane
struct Point
{
    int x, y;
};

/* Following two functions are needed for library function qsort().
   Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */

// Needed to sort array of points according to X coordinate
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

// A utility function to find the distance between two points
float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y) );
}
```

```

// A Brute Force method to return the smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}

// A utility function to find minimum of two float values
float min(float x, float y)
{
    return (x < y)? x : y;
}

// A utility function to find the distance between the closest points of
// strip of given size. All points in strip[] are sorted according to
// y coordinate. They all have an upper bound on minimum distance as d.
// Note that this method seems to be a O(n^2) method, but it's a O(n)
// method as the inner loop runs at most 6 times
float stripClosest(Point strip[], int size, float d)
{
    float min = d; // Initialize the minimum distance as d

    qsort(strip, size, sizeof(Point), compareY);

    // Pick all points one by one and try the next points till the difference
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}

// A recursive function to find the smallest distance. The array P contains
// all points sorted according to x coordinate
float closestUtil(Point P[], int n)
{
    // If there are 2 or 3 points, then use brute force
    if (n <= 3)
        return bruteForce(P, n);

    // Find the middle point
    int mid = n/2;
    Point midPoint = P[mid];

    // Consider the vertical line passing through the middle point
    // calculate the smallest distance dl on left of middle point and
    // dr on right side
    float dl = closestUtil(P, mid);
    float dr = closestUtil(P + mid, n-mid);

    // Find the smaller of two distances
    float d = min(dl, dr);

    // Build an array strip[] that contains points close (closer than d)
    // to the line passing through the middle point
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(P[i].x - midPoint.x) < d)
            strip[j] = P[i], j++;

    // Find the closest points in strip. Return the minimum of d and closest
    // distance is strip[]
    return min(d, stripClosest(strip, j, d) );
}

```

```

// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n)
{
    qsort(P, n, sizeof(Point), compareX);

    // Use recursive function closestUtil() to find the smallest distance
    return closestUtil(P, n);
}

// Driver program to test above functions
int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    printf("The smallest distance is %f ", closest(P, n));
    return 0;
}

```

[Run on IDE](#)

Output:

The smallest distance is 1.414214

Time Complexity Let Time complexity of above algorithm be $T(n)$. Let us assume that we use a $O(n \log n)$ sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in $O(n)$ time, sorts the strip in $O(n \log n)$ time and finally finds the closest points in strip in $O(n)$ time. So $T(n)$ can be expressed as follows

$$T(n) = 2T(n/2) + O(n) + O(n \log n) + O(n)$$

$$T(n) = 2T(n/2) + O(n \log n)$$

$$T(n) = T(n \times \log n \times \log n)$$

Notes

- 1) Time complexity can be improved to $O(n \log n)$ by optimizing step 5 of the above algorithm. We will soon be discussing the optimized solution in a separate post.
- 2) The code finds smallest distance. It can be easily modified to find the points with smallest distance.
- 3) The code uses quick sort which can be $O(n^2)$ in worst case. To have the upper bound as $O(n (\log n)^2)$, a $O(n \log n)$ sorting algorithm like merge sort or heap sort can be used

References:

<http://www.cs.umd.edu/class/fall2013/cmsc451/Lects/lect10.pdf>

<http://www.youtube.com/watch?v=vS4Zn1a9KUc> 

<http://www.youtube.com/watch?v=T3T7T8Ym20M> 

http://en.wikipedia.org/wiki/Closest_pair_of_points_problem



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Divide and Conquer Geometric Closest Pair of Points Divide and Conquer

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

4.3

Average Difficulty : 4.3/5.0
Based on 33 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)

Given two square matrices A and B of size $n \times n$ each, find their multiplication matrix.

Naive Method

Following is a simple way to multiply two matrices.

```
void multiply(int A[][N], int B[][N], int C[][]) {
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

[Run on IDE](#)

Time Complexity of above method is $O(N^3)$.

Divide and Conquer

Following is simple Divide and Conquer method to multiply two square matrices.

- 1) Divide matrices A and B in 4 sub-matrices of size $N/2 \times N/2$ as shown in the below diagram.
- 2) Calculate following values recursively. $ae + bg$, $af + bh$, $ce + dg$ and $cf + dh$.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A B C

A, B and C are square metrices of size $N \times N$

a, b, c and d are submatrices of A, of size $N/2 \times N/2$

e, f, g and h are submatrices of B, of size $N/2 \times N/2$

In the above method, we do 8 multiplications for matrices of size $N/2 \times N/2$ and 4 additions. Addition of two matrices takes $O(N^2)$ time. So the time complexity can be written as

$$T(N) = 8T(N/2) + O(N^2)$$

From [Master's Theorem](#), time complexity of above method is $O(N^3)$
which is unfortunately same as the above naive method.

Simple Divide and Conquer also leads to $O(N^3)$, can there be a better way?

In the above divide and conquer method, the main component for high time complexity is 8 recursive calls. The idea of **Strassen's method** is to reduce the number of recursive calls to 7. Strassen's method is similar to above simple divide and conquer method in the sense that this method also divide matrices to sub-matrices of size $N/2 \times N/2$ as shown in the above diagram, but in Strassen's method, the four sub-matrices of result are calculated using following formulae.

$$\begin{array}{ll} p1 = a(f - h) & p2 = (a + b)h \\ p3 = (c + d)e & p4 = d(g - e) \\ p5 = (a + d)(e + h) & p6 = (b - d)(g + h) \\ p7 = (a - c)(e + f) & \end{array}$$

The $A \times B$ can be calculated using above seven multiplications.

Following are values of four sub-matrices of result C

$$\left[\begin{array}{cc} a & b \\ c & d \end{array} \right] \times \left[\begin{array}{cc} e & f \\ g & h \end{array} \right] = \left[\begin{array}{cc} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{array} \right]$$

A B C

A, B and C are square metrics of size $N \times N$

a, b, c and d are submatrices of A, of size $N/2 \times N/2$

e, f, g and h are submatrices of B, of size $N/2 \times N/2$

p1, p2, p3, p4, p5, p6 and p7 are submatrices of size $N/2 \times N/2$

Time Complexity of Strassen's Method

Addition and Subtraction of two matrices takes $O(N^2)$ time. So time complexity can be written as

$$T(N) = 7T(N/2) + O(N^2)$$

From [Master's Theorem](#), time complexity of above method is $O(N^{\log 7})$ which is approximately $O(N^{2.8074})$

Generally Strassen's Method is not preferred for practical applications for following reasons.

- 1) The constants used in Strassen's method are high and for a typical application Naive method works better.
- 2) For Sparse matrices, there are better methods especially designed for them.
- 3) The submatrices in recursion take extra space.
- 4) Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen's algorithm than in Naive Method (Source: [CLRS Book](#))

References:

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

<https://www.youtube.com/watch?v=LOLebQ8nKHA> 

<https://www.youtube.com/watch?v=QXY4RskLQcl> 

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner Company Wise Coding Practice

Divide and Conquer

Divide and Conquer

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

3.4

Average Difficulty : 3.4/5.0
Based on 15 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

HANDBOOK OF ALGORITHMS

Section
Backtracking

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 1 (The Knight's tour problem)

Backtracking is an algorithmic paradigm that tries different solutions until finds a solution that “works”. Problems which are typically solved using backtracking technique have following property in common. These problems can only be solved by trying every possible configuration and each configuration is tried only once. A Naive solution for these problems is to try all configurations and output a configuration that follows given problem constraints. Backtracking works in incremental way and is an optimization over the Naive solution where all possible configurations are generated and tried.

For example, consider the following [Knight's Tour](#) problem.

The knight is placed on the first block of an empty board and, moving according to the rules of chess, must visit each square exactly once.

Path followed by Knight to cover all the cells

0	59	38	33	30	17	8	63
37	34	31	60	9	62	29	16
58	1	36	39	32	27	18	7
35	48	41	26	61	10	15	28
42	57	2	49	40	23	6	19
47	50	45	54	25	20	11	14
56	43	52	3	22	13	24	5
51	46	55	44	53	4	21	12

Let us first discuss the Naive algorithm for this problem and then the Backtracking algorithm.

Naive Algorithm for Knight's tour

The Naive Algorithm is to generate all tours one by one and check if the generated tour satisfies the constraints.

```

while there are untried tours
{
    generate the next tour
    if this tour covers all squares
    {
        print this path;
    }
}

```

Backtracking works in an incremental way to attack problems. Typically, we start from an empty solution vector and one by one add items (Meaning of item varies from problem to problem. In context of Knight's tour problem, an item is a Knight's move). When we add an item, we check if adding the current item violates the problem constraint, if it does then we remove the item and try other alternatives. If none of the alternatives work out then we go to previous stage and remove the item added in the previous stage. If we reach the initial stage back then we say that no solution exists. If adding an item doesn't violate constraints then we recursively add items one by one. If the solution vector becomes complete then we print the solution.

Backtracking Algorithm for Knight's tour

Following is the Backtracking algorithm for Knight's tour problem.

```
If all squares are visited
    print the solution
Else
    a) Add one of the next moves to solution vector and recursively
       check if this move leads to a solution. (A Knight can make maximum
       eight moves. We choose one of the 8 moves in this step).
    b) If the move chosen in the above step doesn't lead to a solution
       then remove this move from the solution vector and try other
       alternative moves.
    c) If none of the alternatives work then return false (Returning false
       will remove the previously added item in recursion and if false is
       returned by the initial call of recursion then "no solution exists" )
```

Following are implementations for Knight's tour problem. It prints one of the possible solutions in 2D matrix form. Basically, the output is a 2D 8*8 matrix with numbers from 0 to 63 and these numbers show steps made by Knight.

```
// C program for Knight Tour problem
#include<stdio.h>
#define N 8

int solveKTUtil(int x, int y, int movei, int sol[N][N],
                int xMove[], int yMove[]);

/* A utility function to check if i,j are valid indexes
   for N*N chessboard */
bool isSafe(int x, int y, int sol[N][N])
{
    return ( x >= 0 && x < N && y >= 0 &&
             y < N && sol[x][y] == -1);
}

/* A utility function to print solution matrix sol[N][N] */
void printSolution(int sol[N][N])
{
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < N; y++)
            printf(" %2d ", sol[x][y]);
        printf("\n");
    }
}

/* This function solves the Knight Tour problem using
   Backtracking. This function mainly uses solveKTUtil()
   to solve the problem. It returns false if no complete
   tour is possible, otherwise return true and prints the
   tour.
   Please note that there may be more than one solutions,
   this function prints one of the feasible solutions. */

```

```

bool solveKT()
{
    int sol[N][N];

    /* Initialization of solution matrix */
    for (int x = 0; x < N; x++)
        for (int y = 0; y < N; y++)
            sol[x][y] = -1;

    /* xMove[] and yMove[] define next move of Knight.
       xMove[] is for next value of x coordinate
       yMove[] is for next value of y coordinate */
    int xMove[8] = { 2, 1, -1, -2, -2, -1, 1, 2 };
    int yMove[8] = { 1, 2, 2, 1, -1, -2, -2, -1 };

    // Since the Knight is initially at the first block
    sol[0][0] = 0;

    /* Start from 0,0 and explore all tours using
       solveKTUtil() */
    if (solveKTUtil(0, 0, 1, sol, xMove, yMove) == false)
    {
        printf("Solution does not exist");
        return false;
    }
    else
        printSolution(sol);

    return true;
}

/* A recursive utility function to solve Knight Tour
   problem */
int solveKTUtil(int x, int y, int movei, int sol[N][N],
                int xMove[N], int yMove[N])
{
    int k, next_x, next_y;
    if (movei == N*N)
        return true;

    /* Try all next moves from the current coordinate x, y */
    for (k = 0; k < 8; k++)
    {
        next_x = x + xMove[k];
        next_y = y + yMove[k];
        if (isSafe(next_x, next_y, sol))
        {
            sol[next_x][next_y] = movei;
            if (solveKTUtil(next_x, next_y, movei+1, sol,
                           xMove, yMove) == true)
                return true;
            else
                sol[next_x][next_y] = -1;// backtracking
        }
    }

    return false;
}

/* Driver program to test above functions */
int main()
{
    solveKT();
    return 0;
}

```

[Run on IDE](#)

Java

```
// Java program for Knight Tour problem
```

```

class KnightTour {
    static int N = 8;

    /* A utility function to check if i,j are
       valid indexes for N*N chessboard */
    static boolean isSafe(int x, int y, int sol[][][]) {
        return (x >= 0 && x < N && y >= 0 &&
                y < N && sol[x][y] == -1);
    }

    /* A utility function to print solution
       matrix sol[N][N] */
    static void printSolution(int sol[][][]) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++)
                System.out.print(sol[x][y] + " ");
            System.out.println();
        }
    }

    /* This function solves the Knight Tour problem
       using Backtracking. This function mainly
       uses solveKTUtil() to solve the problem. It
       returns false if no complete tour is possible,
       otherwise return true and prints the tour.
       Please note that there may be more than one
       solutions, this function prints one of the
       feasible solutions. */
    static boolean solveKT() {
        int sol[][] = new int[8][8];

        /* Initialization of solution matrix */
        for (int x = 0; x < N; x++)
            for (int y = 0; y < N; y++)
                sol[x][y] = -1;

        /* xMove[] and yMove[] define next move of Knight.
           xMove[] is for next value of x coordinate
           yMove[] is for next value of y coordinate */
        int xMove[] = {2, 1, -1, -2, -2, -1, 1, 2};
        int yMove[] = {1, 2, 2, 1, -1, -2, -2, -1};

        // Since the Knight is initially at the first block
        sol[0][0] = 0;

        /* Start from 0,0 and explore all tours using
           solveKTUtil() */
        if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {
            System.out.println("Solution does not exist");
            return false;
        } else
            printSolution(sol);

        return true;
    }

    /* A recursive utility function to solve Knight
       Tour problem */
    static boolean solveKTUtil(int x, int y, int movei,
                               int sol[][], int xMove[],
                               int yMove[]) {
        int k, next_x, next_y;
        if (movei == N * N)
            return true;

        /* Try all next moves from the current coordinate
           x, y */
        for (k = 0; k < 8; k++) {
            next_x = x + xMove[k];
            next_y = y + yMove[k];
            if (isSafe(next_x, next_y, sol)) {
                sol[next_x][next_y] = movei;
                if (solveKTUtil(next_x, next_y, movei + 1,
                                sol, xMove, yMove))

```

```

        return true;
    else
        sol[next_x][next_y] = -1; // backtracking
    }
}

return false;
}

/* Driver program to test above functions */
public static void main(String args[]) {
    solveKT();
}
// This code is contributed by Abhishek Shankhadhar

```

[Run on IDE](#)

Output:

```

0 59 38 33 30 17 8 63
37 34 31 60 9 62 29 16
58 1 36 39 32 27 18 7
35 48 41 26 61 10 15 28
42 57 2 49 40 23 6 19
47 50 45 54 25 20 11 14
56 43 52 3 22 13 24 5
51 46 55 44 53 4 21 12

```

Note that Backtracking is not the best solution for the Knight's tour problem. See [this](#) for other better solutions. The purpose of this post is to explain Backtracking with an example.

References:

<http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf>
<http://www.cis.upenn.edu/~matuszek/cit594-2009/Lectures/35-backtracking.ppt>
<http://mathworld.wolfram.com/KnightsTour.html>
http://en.wikipedia.org/wiki/Knight%27s_tour

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Kerala Packages

Kerala tour packages @
cheap rate Kerala Hotels @
lowest cost

kochiholidays.com



GATE CS Corner Company Wise Coding Practice

Backtracking Backtracking

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

3.6

Average Difficulty : 3.6/5.0
Based on 50 vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 2 (Rat in a Maze)

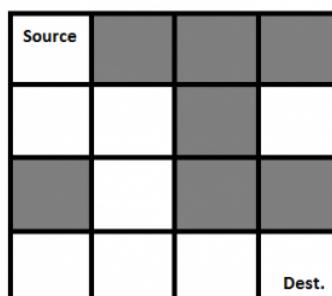
We have discussed Backtracking and Knight's tour problem in [Set 1](#). Let us discuss Rat in a [Maze](#) as another example problem that can be solved using Backtracking.

A Maze is given as $N \times N$ binary matrix of blocks where source block is the upper left most block i.e., $\text{maze}[0][0]$ and destination block is lower rightmost block i.e., $\text{maze}[N-1][N-1]$. A rat starts from source and has to reach destination. The rat can move only in two directions: forward and down.

In the maze matrix, 0 means the block is dead end and 1 means the block can be used in the path from source to destination. Note that this is a simple version of the typical Maze problem. For example, a more complex version can be that the rat can move in 4 directions and a more complex version can be with limited number of moves.

Following is an example maze.

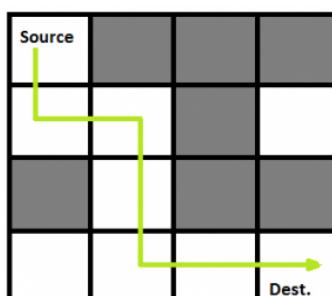
Gray blocks are dead ends (value = 0).



Following is binary matrix representation of the above maze.

```
{1, 0, 0, 0}
{1, 1, 0, 1}
{0, 1, 0, 0}
{1, 1, 1, 1}
```

Following is maze with highlighted solution path.



Following is the solution matrix (output of program) for the above input matrix.

```
{1, 0, 0, 0}
{1, 1, 0, 0}
{0, 1, 0, 0}
{0, 1, 1, 1}
```

All entries in solution path are marked as 1.

We strongly recommend that you click here and practice it, before moving on to the solution.

Naive Algorithm

The Naive Algorithm is to generate all paths from source to destination and one by one check if the generated path satisfies the constraints.

```
while there are untried paths
{
    generate the next path
    if this path has all blocks as 1
    {
        print this path;
    }
}
```

Backtracking Algorithm

```
If destination is reached
    print the solution matrix
Else
    a) Mark current cell in solution matrix as 1.
    b) Move forward in horizontal direction and recursively check if this
       move leads to a solution.
    c) If the move chosen in the above step doesn't lead to a solution
       then move down and check if this move leads to a solution.
    d) If none of the above solutions work then unmark this cell as 0
       (BACKTRACK) and return false.
```

Implementation of Backtracking solution

```
/* C/C++ program to solve Rat in a Maze problem using
   backtracking */
#include<stdio.h>

// Maze size
#define N 4

bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]);

/* A utility function to print solution matrix sol[N][N] */
void printSolution(int sol[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
    }
```

```

        printf(" %d ", sol[i][j]);
    printf("\n");
}

/* A utility function to check if x,y is valid index for N*N maze */
bool isSafe(int maze[N][N], int x, int y)
{
    // if (x,y outside maze) return false
    if(x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1)
        return true;

    return false;
}

/* This function solves the Maze problem using Backtracking. It mainly
   uses solveMazeUtil() to solve the problem. It returns false if no
   path is possible, otherwise return true and prints the path in the
   form of 1s. Please note that there may be more than one solutions,
   this function prints one of the feasible solutions.*/
bool solveMaze(int maze[N][N])
{
    int sol[N][N] = { {0, 0, 0, 0},
                      {0, 0, 0, 0},
                      {0, 0, 0, 0},
                      {0, 0, 0, 0}
                    };

    if(solveMazeUtil(maze, 0, 0, sol) == false)
    {
        printf("Solution doesn't exist");
        return false;
    }

    printSolution(sol);
    return true;
}

/* A recursive utility function to solve Maze problem */
bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])
{
    // if (x,y is goal) return true
    if(x == N-1 && y == N-1)
    {
        sol[x][y] = 1;
        return true;
    }

    // Check if maze[x][y] is valid
    if(isSafe(maze, x, y) == true)
    {
        // mark x,y as part of solution path
        sol[x][y] = 1;

        /* Move forward in x direction */
        if (solveMazeUtil(maze, x+1, y, sol) == true)
            return true;

        /* If moving in x direction doesn't give solution then
           Move down in y direction */
        if (solveMazeUtil(maze, x, y+1, sol) == true)
            return true;

        /* If none of the above movements work then BACKTRACK:
           unmark x,y as part of solution path */
        sol[x][y] = 0;
        return false;
    }

    return false;
}

// driver program to test above function
int main()

```

```

{
    int maze[N][N] = { {1, 0, 0, 0},
                      {1, 1, 0, 1},
                      {0, 1, 0, 0},
                      {1, 1, 1, 1}
                    };

    solveMaze(maze);
    return 0;
}

```

[Run on IDE](#)

Java

```

/* Java program to solve Rat in a Maze problem using
backtracking */

public class RatMaze
{
    final int N = 4;

    /* A utility function to print solution matrix
       sol[N][N] */
    void printSolution(int sol[][])
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                System.out.print(" " + sol[i][j] + " ");
            System.out.println();
        }
    }

    /* A utility function to check if x,y is valid
       index for N*N maze */
    boolean isSafe(int maze[][], int x, int y)
    {
        // if (x,y outside maze) return false
        return (x >= 0 && x < N && y >= 0 &&
               y < N && maze[x][y] == 1);
    }

    /* This function solves the Maze problem using
       Backtracking. It mainly uses solveMazeUtil()
       to solve the problem. It returns false if no
       path is possible, otherwise return true and
       prints the path in the form of 1s. Please note
       that there may be more than one solutions, this
       function prints one of the feasible solutions.*/
    boolean solveMaze(int maze[][])
    {
        int sol[][] = {{0, 0, 0, 0},
                      {0, 0, 0, 0},
                      {0, 0, 0, 0},
                      {0, 0, 0, 0}
                    };

        if (solveMazeUtil(maze, 0, 0, sol) == false)
        {
            System.out.print("Solution doesn't exist");
            return false;
        }

        printSolution(sol);
        return true;
    }

    /* A recursive utility function to solve Maze
       problem */

```

```

boolean solveMazeUtil(int maze[][], int x, int y,
                      int sol[][])
{
    // if (x,y is goal) return true
    if (x == N - 1 && y == N - 1)
    {
        sol[x][y] = 1;
        return true;
    }

    // Check if maze[x][y] is valid
    if (isSafe(maze, x, y) == true)
    {
        // mark x,y as part of solution path
        sol[x][y] = 1;

        /* Move forward in x direction */
        if (solveMazeUtil(maze, x + 1, y, sol))
            return true;

        /* If moving in x direction doesn't give
           solution then Move down in y direction */
        if (solveMazeUtil(maze, x, y + 1, sol))
            return true;

        /* If none of the above movements work then
           BACKTRACK: unmark x,y as part of solution
           path */
        sol[x][y] = 0;
        return false;
    }

    return false;
}

public static void main(String args[])
{
    RatMaze rat = new RatMaze();
    int maze[][] = {{1, 0, 0, 0},
                    {1, 1, 0, 1},
                    {0, 1, 0, 0},
                    {1, 1, 1, 1}};
    rat.solveMaze(maze);
}
// This code is contributed by Abhishek Shankhadhar

```

[Run on IDE](#)

Output: The 1 values show the path for rat

```

1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1

```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Backtracking](#) [Backtracking](#)

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

3.2

Average Difficulty : **3.2/5.0**
Based on **59** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

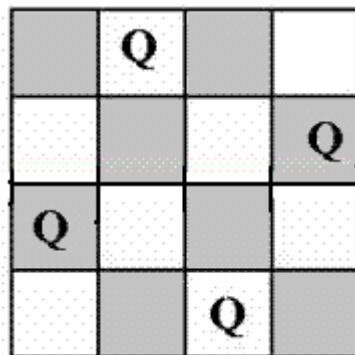


[Login/Register](#)

Backtracking | Set 3 (N Queen Problem)

We have discussed Knight's tour and Rat in a Maze problems in [Set 1](#) and [Set 2](#) respectively. Let us discuss N Queen as another example problem that can be solved using Backtracking.

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example following is the output matrix for above 4 queen solution.

```
{ 0,  1,  0,  0}
{ 0,  0,  0,  1}
{ 1,  0,  0,  0}
{ 0,  0,  1,  0}
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Naive Algorithm

Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints.

```
while there are untried configurations
{
    generate the next configuration
    if queens don't attack in this configuration then
    {
        print this configuration;
```

```

    }
}

```

Backtracking Algorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

- 1) Start in the leftmost column
- 2) If all queens are placed
 - return true
- 3) Try all rows in the current column. Do following for every tried row.
 - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
 - b) If placing queen in [row, column] leads to a solution then return true.
 - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- 3) If all rows have been tried and nothing worked, return false to trigger backtracking.

Implementation of Backtracking solution

```

/* C/C++ program to solve N Queen Problem using
   backtracking */
#define N 4
#include<stdio.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}

/* A utility function to check if a queen can
   be placed on board[row][col]. Note that this
   function is called when "col" queens are
   already placed in columns from 0 to col -1.
   So we need to check only left side for
   attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;

```

```

/* Check lower diagonal on left side */
for (i=row, j=col; j>=0 && i<N; i++, j--)
    if (board[i][j])
        return false;

return true;
}

/* A recursive utility function to solve N
Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: If all queens are placed
       then return true */
    if (col >= N)
        return true;

    /* Consider this column and try placing
       this queen in all rows one by one */
    for (int i = 0; i < N; i++)
    {
        /* Check if queen can be placed on
           board[i][col] */
        if (isSafe(board, i, col))
        {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            if (solveNQUtil(board, col + 1))
                return true;

            /* If placing queen in board[i][col]
               doesn't lead to a solution, then
               remove queen from board[i][col] */
            board[i][col] = 0; // BACKTRACK
        }
    }

    /* If queen can not be place in any row in
       this colum col then return false */
    return false;
}

/* This function solves the N Queen problem using
Backtracking. It mainly uses solveNQUtil() to
solve the problem. It returns false if queens
cannot be placed, otherwise return true and
prints placement of queens in the form of 1s.
Please note that there may be more than one
solutions, this function prints one of the
feasible solutions.*/
bool solveNQ()
{
    int board[N][N] = { {0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0}
    };

    if (solveNQUtil(board, 0) == false)
    {
        printf("Solution does not exist");
        return false;
    }

    printSolution(board);
    return true;
}

// driver program to test above function
int main()
{
    solveNQ();
}

```

```

    return 0;
}

```

[Run on IDE](#)

Java

```

/* Java program to solve N Queen Problem using
backtracking */
public class NQueenProblem
{
    final int N = 4;

    /* A utility function to print solution */
    void printSolution(int board[][])
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                System.out.print(" " + board[i][j]
                                + " ");
            System.out.println();
        }
    }

    /* A utility function to check if a queen can
    be placed on board[row][col]. Note that this
    function is called when "col" queens are already
    placed in columns from 0 to col -1. So we need
    to check only left side for attacking queens */
    boolean isSafe(int board[][], int row, int col)
    {
        int i, j;

        /* Check this row on left side */
        for (i = 0; i < col; i++)
            if (board[row][i] == 1)
                return false;

        /* Check upper diagonal on left side */
        for (i=row, j=col; i>=0 && j>=0; i--, j--)
            if (board[i][j] == 1)
                return false;

        /* Check lower diagonal on left side */
        for (i=row, j=col; j>=0 && i<N; i++, j--)
            if (board[i][j] == 1)
                return false;

        return true;
    }

    /* A recursive utility function to solve N
    Queen problem */
    boolean solveNQUtil(int board[][], int col)
    {
        /* base case: If all queens are placed
        then return true */
        if (col >= N)
            return true;

        /* Consider this column and try placing
        this queen in all rows one by one */
        for (int i = 0; i < N; i++)
        {
            /* Check if queen can be placed on
            board[i][col] */
            if (isSafe(board, i, col))
            {
                /* Place this queen in board[i][col] */
                board[i][col] = 1;

```

```

        /* recur to place rest of the queens */
        if (solveNQUtil(board, col + 1) == true)
            return true;

        /* If placing queen in board[i][col]
           doesn't lead to a solution then
           remove queen from board[i][col] */
        board[i][col] = 0; // BACKTRACK
    }

    /* If queen can not be placed in any row in
       this column col, then return false */
    return false;
}

/* This function solves the N Queen problem using
   Backtracking. It mainly uses solveNQUtil() to
   solve the problem. It returns false if queens
   cannot be placed, otherwise return true and
   prints placement of queens in the form of 1s.
   Please note that there may be more than one
   solutions, this function prints one of the
   feasible solutions.*/
boolean solveNQ()
{
    int board[][] = {{0, 0, 0, 0},
                    {0, 0, 0, 0},
                    {0, 0, 0, 0},
                    {0, 0, 0, 0}};
};

if (solveNQUtil(board, 0) == false)
{
    System.out.print("Solution does not exist");
    return false;
}

printSolution(board);
return true;
}

// driver program to test above function
public static void main(String args[])
{
    NQueenProblem Queen = new NQueenProblem();
    Queen.solveNQ();
}
}
// This code is contributed by Abhishek Shankhadhar

```

[Run on IDE](#)

Output: The 1 values indicate placements of queens

```

0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

```

Sources:

- <http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf>
- http://en.literateprograms.org/Eight_queens_puzzle_%28C%29
- http://en.wikipedia.org/wiki/Eight_queens_puzzle

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Backtracking](#) [Backtracking](#)

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 60 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 4 (Subset Sum)

Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented).

We strongly recommend that you click here and practice it, before moving on to the solution.

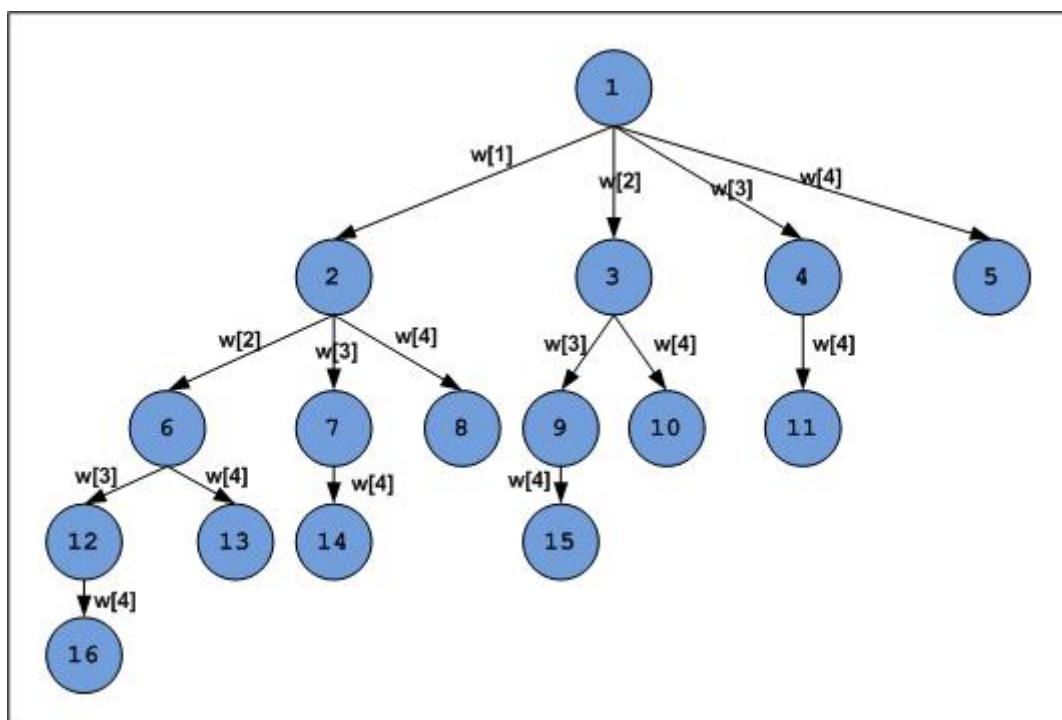
Exhaustive Search Algorithm for Subset Sum

One way to find subsets that sum to K is to consider all possible subsets. A **power set** contains all those subsets generated from a given set. The size of such a power set is 2^N .

Backtracking Algorithm for Subset Sum

Using exhaustive search we consider all subsets irrespective of whether they satisfy given constraints or not. Backtracking can be used to make a systematic consideration of the elements to be selected.

Assume given set of 4 elements, say $w[1] \dots w[4]$. Tree diagrams can be used to design backtracking algorithms. The following tree diagram depicts approach of generating variable sized tuple.



In the above tree, a node represents function call and a branch represents candidate element. The root node contains 4 children. In other words, root considers every element of the set as different branch. The next level subtrees correspond to the subsets that includes the parent node. The branches at each level represent tuple element to be considered. For example, if we are at level 1, `tuple_vector[1]` can take any value of four branches generated. If we are at level 2 of left most node, `tuple_vector[2]` can take any value of three branches generated, and so on...

For example the left most child of root generates all those subsets that include $w[1]$. Similarly the second child of root generates all those subsets that includes $w[2]$ and excludes $w[1]$.

As we go down along depth of tree we add elements so far, and if the added sum is satisfying explicit constraints, we will continue to generate child nodes further. Whenever the constraints are not met, we stop further generation of sub-trees of that node, and backtrack to previous node to explore the nodes not yet explored. In many scenarios, it saves considerable amount of processing time.

The tree should trigger a clue to implement the backtracking algorithm (try yourself). It prints all those subsets whose sum add up to given number. We need to explore the nodes along the breadth and depth of the tree. Generating nodes along breadth is controlled by loop and nodes along the depth are generated using recursion (post order traversal). Pseudo code given below,

```

if(subset is satisfying the constraint)
    print the subset
    exclude the current element and consider next element
else
    generate the nodes of present level along breadth of tree and
    recur for next levels
  
```

Following is C implementation of subset sum using variable size tuple vector. Note that the following program explores all possibilities similar to exhaustive search. It is to demonstrate how backtracking can be used. See next code to verify, how we can optimize the backtracking solution.

```

#include <stdio.h>
#include <stdlib.h>

#define ARRSIZE(a) (sizeof(a))/(sizeof(a[0]))

static int total_nodes;
// prints subset found
void printSubset(int A[], int size)
{
    for(int i = 0; i < size; i++)
    {
        printf("%*d", 5, A[i]);
    }

    printf("\n");
}

// inputs
// s          - set vector
// t          - tuplet vector
// s_size     - set size
// t_size     - tuplet size so far
// sum        - sum so far
// ite        - nodes count
// target_sum - sum to be found
void subset_sum(int s[], int t[],
                int s_size, int t_size,
                int sum, int ite,
                int const target_sum)
{
    total_nodes++;
    if( target_sum == sum )
  
```

```

    {
        // We found subset
        printSubset(t, t_size);
        // Exclude previously added item and consider next candidate
        subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
        return;
    }
    else
    {
        // generate nodes along the breadth
        for( int i = ite; i < s_size; i++ )
        {
            t[t_size] = s[i];
            // consider next level node (along depth)
            subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
        }
    }
}

// Wrapper to print subsets that sum to target_sum
// input is weights vector and target_sum
void generateSubsets(int s[], int size, int target_sum)
{
    int *tuplet_vector = (int *)malloc(sizeof(int) * size);
    subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
    free(tuplet_vector);
}

int main()
{
    int weights[] = {10, 7, 5, 18, 12, 20, 15};
    int size = ARRSIZE(weights);

    generateSubsets(weights, size, 35);
    printf("Nodes generated %d\n", total_nodes);
    return 0;
}

```

Run on IDE

The power of backtracking appears when we combine explicit and implicit constraints, and we stop generating nodes when these checks fail. We can improve the above algorithm by strengthening the constraint checks and presorting the data. By sorting the initial array, we need not to consider rest of the array, once the sum so far is greater than target number. We can backtrack and check other possibilities.

Similarly, assume the array is presorted and we found one subset. We can generate next node excluding the present node only when inclusion of next node satisfies the constraints. Given below is optimized implementation (it prunes the subtree if it is not satisfying contraints).

```

#include <stdio.h>
#include <stdlib.h>

#define ARRSIZE(a) (sizeof(a))/(sizeof(a[0]))

static int total_nodes;

// prints subset found
void printSubset(int A[], int size)
{
    for(int i = 0; i < size; i++)
    {
        printf("%*d", 5, A[i]);
    }

    printf("\n");
}

// qsort compare function

```

```

int comparator(const void *pLhs, const void *pRhs)
{
    int *lhs = (int *)pLhs;
    int *rhs = (int *)pRhs;

    return *lhs > *rhs;
}

// inputs
// s           - set vector
// t           - tuplet vector
// s_size      - set size
// t_size      - tuplet size so far
// sum         - sum so far
// ite         - nodes count
// target_sum  - sum to be found
void subset_sum(int s[], int t[],
                int s_size, int t_size,
                int sum, int ite,
                int const target_sum)
{
    total_nodes++;

    if( target_sum == sum )
    {
        // We found sum
        printSubset(t, t_size);

        // constraint check
        if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
        {
            // Exclude previous added item and consider next candidate
            subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
        }
        return;
    }
    else
    {
        // constraint check
        if( ite < s_size && sum + s[ite] <= target_sum )
        {
            // generate nodes along the breadth
            for( int i = ite; i < s_size; i++ )
            {
                t[t_size] = s[i];

                if( sum + s[i] <= target_sum )
                {
                    // consider next level node (along depth)
                    subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
                }
            }
        }
    }
}

// Wrapper that prints subsets that sum to target_sum
void generateSubsets(int s[], int size, int target_sum)
{
    int *tuplet_vector = (int *)malloc(size * sizeof(int));

    int total = 0;

    // sort the set
    qsort(s, size, sizeof(int), &comparator);

    for( int i = 0; i < size; i++ )
    {
        total += s[i];
    }

    if( s[0] <= target_sum && total >= target_sum )
    {

```

```

        subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);

    }

    free(tuplet_vector);
}

int main()
{
    int weights[] = {15, 22, 14, 26, 32, 9, 16, 8};
    int target = 53;

    int size = ARRSIZE(weights);
    generateSubsets(weights, size, target);
    printf("Nodes generated %d\n", total_nodes);
    return 0;
}

```

[Run on IDE](#)

As another approach, we can generate the tree in fixed size tuple analogs to binary pattern. We will kill the subtrees when the constraints are not satisfied.

-- **Venki**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Backtracking](#) [Adobe-Question](#) [Backtracking](#)

About Venki

Software Engineer

[View all posts by Venki →](#)

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

4.3

Average Difficulty : **4.3/5.0**
Based on **25** vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 5 (m Coloring Problem)

Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with same color. Here coloring of a graph means assignment of colors to all vertices.

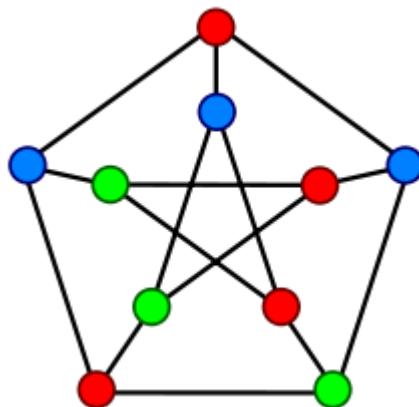
Input:

- 1) A 2D array $\text{graph}[V][V]$ where V is the number of vertices in graph and $\text{graph}[V][V]$ is adjacency matrix representation of the graph. A value $\text{graph}[i][j]$ is 1 if there is a direct edge from i to j, otherwise $\text{graph}[i][j]$ is 0.
- 2) An integer m which is maximum number of colors that can be used.

Output:

An array $\text{color}[V]$ that should have numbers from 1 to m. $\text{color}[i]$ should represent the color assigned to the ith vertex. The code should also return false if the graph cannot be colored with m colors.

Following is an example graph (from [Wiki page](#)) that can be colored with 3 colors.



We strongly recommend that you click here and practice it, before moving on to the solution.

Naive Algorithm

Generate all possible configurations of colors and print a configuration that satisfies the given constraints.

```

while there are untried configurations
{
    generate the next configuration
    if no adjacent vertices are colored with same color
    {
        print this configuration;
    }
}

```

```

    }
}

```

There will be V^m configurations of colors.

Backtracking Algorithm

The idea is to assign colors one by one to different vertices, starting from the vertex 0. Before assigning a color, we check for safety by considering already assigned colors to the adjacent vertices. If we find a color assignment which is safe, we mark the color assignment as part of solution. If we do not find a color due to clashes then we backtrack and return false.

Implementation of Backtracking solution

```

#include<stdio.h>

// Number of vertices in the graph
#define V 4

void printSolution(int color[]);

/* A utility function to check if the current color assignment
   is safe for vertex v */
bool isSafe (int v, bool graph[V][V], int color[], int c)
{
    for (int i = 0; i < V; i++)
        if (graph[v][i] && c == color[i])
            return false;
    return true;
}

/* A recursive utility function to solve m coloring problem */
bool graphColoringUtil(bool graph[V][V], int m, int color[], int v)
{
    /* base case: If all vertices are assigned a color then
       return true */
    if (v == V)
        return true;

    /* Consider this vertex v and try different colors */
    for (int c = 1; c <= m; c++)
    {
        /* Check if assignment of color c to v is fine*/
        if (isSafe(v, graph, color, c))
        {
            color[v] = c;

            /* recur to assign colors to rest of the vertices */
            if (graphColoringUtil (graph, m, color, v+1) == true)
                return true;

            /* If assigning color c doesn't lead to a solution
               then remove it */
            color[v] = 0;
        }
    }

    /* If no color can be assigned to this vertex then return false */
    return false;
}

/* This function solves the m Coloring problem using Backtracking.
It mainly uses graphColoringUtil() to solve the problem. It returns
false if the m colors cannot be assigned, otherwise return true and
prints assignments of colors to all vertices. Please note that there
may be more than one solutions, this function prints one of the
feasible solutions.*/

```

```

bool graphColoring(bool graph[V][V], int m)
{
    // Initialize all color values as 0. This initialization is needed
    // correct functioning of isSafe()
    int *color = new int[V];
    for (int i = 0; i < V; i++)
        color[i] = 0;

    // Call graphColoringUtil() for vertex 0
    if (graphColoringUtil(graph, m, color, 0) == false)
    {
        printf("Solution does not exist");
        return false;
    }

    // Print the solution
    printSolution(color);
    return true;
}

/* A utility function to print solution */
void printSolution(int color[])
{
    printf("Solution Exists:");
    " Following are the assigned colors \n");
    for (int i = 0; i < V; i++)
        printf(" %d ", color[i]);
    printf("\n");
}

// driver program to test above function
int main()
{
    /* Create following graph and test whether it is 3 colorable
       (3)---(2)
          |
          |
          |
          (0)---(1)
    */
    bool graph[V][V] = {{0, 1, 1, 1},
                        {1, 0, 1, 0},
                        {1, 1, 0, 1},
                        {1, 0, 1, 0},
    };
    int m = 3; // Number of colors
    graphColoring (graph, m);
    return 0;
}

```

[Run on IDE](#)

Java

```

/* Java program for solution of M Coloring problem
   using backtracking */
public class mColoringProblem {
    final int V = 4;
    int color[];

    /* A utility function to check if the current
       color assignment is safe for vertex v */
    boolean isSafe(int v, int graph[][], int color[],
                  int c)
    {
        for (int i = 0; i < V; i++)
            if (graph[v][i] == 1 && c == color[i])
                return false;
        return true;
    }
}

```

```

/* A recursive utility function to solve m
   coloring problem */
boolean graphColoringUtil(int graph[][][], int m,
                           int color[], int v)
{
    /* base case: If all vertices are assigned
       a color then return true */
    if (v == V)
        return true;

    /* Consider this vertex v and try different
       colors */
    for (int c = 1; c <= m; c++)
    {
        /* Check if assignment of color c to v
           is fine*/
        if (isSafe(v, graph, color, c))
        {
            color[v] = c;

            /* recur to assign colors to rest
               of the vertices */
            if (graphColoringUtil(graph, m,
                                  color, v + 1))
                return true;

            /* If assigning color c doesn't lead
               to a solution then remove it */
            color[v] = 0;
        }
    }

    /* If no color can be assigned to this vertex
       then return false */
    return false;
}

/* This function solves the m Coloring problem using
   Backtracking. It mainly uses graphColoringUtil()
   to solve the problem. It returns false if the m
   colors cannot be assigned, otherwise return true
   and prints assignments of colors to all vertices.
   Please note that there may be more than one
   solutions, this function prints one of the
   feasible solutions.*/
boolean graphColoring(int graph[][][], int m)
{
    // Initialize all color values as 0. This
    // initialization is needed correct functioning
    // of isSafe()
    color = new int[V];
    for (int i = 0; i < V; i++)
        color[i] = 0;

    // Call graphColoringUtil() for vertex 0
    if (!graphColoringUtil(graph, m, color, 0))
    {
        System.out.println("Solution does not exist");
        return false;
    }

    // Print the solution
    printSolution(color);
    return true;
}

/* A utility function to print solution */
void printSolution(int color[])
{
    System.out.println("Solution Exists: Following" +
                       " are the assigned colors");
    for (int i = 0; i < V; i++)
        System.out.print(" " + color[i] + " ");
    System.out.println();
}

```

```

}

// driver program to test above function
public static void main(String args[])
{
    mColoringProblem Coloring = new mColoringProblem();
    /* Create following graph and test whether it is
       3 colorable
       (3)---(2)
           |
           |
           |
           (0)---(1)
    */
    int graph[][][] = {{0, 1, 1, 1},
                       {1, 0, 1, 0},
                       {1, 1, 0, 1},
                       {1, 0, 1, 0},
    };
    int m = 3; // Number of colors
    Coloring.graphColoring(graph, m);
}
// This code is contributed by Abhishek Shankhadhar
}

```

[Run on IDE](#)

Output:

Solution Exists: Following are the assigned colors
 1 2 3 2

References:

http://en.wikipedia.org/wiki/Graph_coloring

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

The advertisement features a large image of a modern building with a sign that reads "LOLA CERAMICS (M) SDN. BHD." and "BY LOLA GROUP". Below the building is a smaller image of a group of men standing outdoors. The text "China huge tile factory" is displayed in an orange bar at the bottom left. At the bottom right is a circular orange button with a white arrow pointing right. The text "Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia" is also present.

GATE CS Corner Company Wise Coding Practice

[Backtracking](#) [Backtracking](#)

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **34** vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 6 (Hamiltonian Cycle)

Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then print the path. Following are the input and output of the required function.

Input:

A 2D array $\text{graph}[V][V]$ where V is the number of vertices in graph and $\text{graph}[V][V]$ is adjacency matrix representation of the graph. A value $\text{graph}[i][j]$ is 1 if there is a direct edge from i to j , otherwise $\text{graph}[i][j]$ is 0.

Output:

An array $\text{path}[V]$ that should contain the Hamiltonian Path. $\text{path}[i]$ should represent the i th vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

For example, a Hamiltonian Cycle in the following graph is $\{0, 1, 2, 4, 3, 0\}$. There are more Hamiltonian Cycles in the graph like $\{0, 3, 4, 2, 1, 0\}$

```
(0)--(1)--(2)
 |   / \   |
 |   /   \   |
 | /       \ |
 (3)-----(4)
```

And the following graph doesn't contain any Hamiltonian Cycle.

```
(0)--(1)--(2)
 |   / \   |
 |   /   \   |
 | /       \ |
 (3)     (4)
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Naive Algorithm

Generate all possible configurations of vertices and print a configuration that satisfies the given constraints. There will be $n!$ (n factorial) configurations.

```

while there are untried configurations
{
    generate the next configuration
    if ( there are edges between two consecutive vertices of this
        configuration and there is an edge from the last vertex to
        the first ).

    {
        print this configuration;
        break;
    }
}

```

Backtracking Algorithm

Create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return false.

Implementation of Backtracking solution

Following are implementations of the Backtracking solution.

```

/* C/C++ program for solution of Hamiltonian Cycle problem
   using backtracking */
#include<stdio.h>

// Number of vertices in the graph
#define V 5

void printSolution(int path[]);

/* A utility function to check if the vertex v can be added at
   index 'pos' in the Hamiltonian Cycle constructed so far (stored
   in 'path[]') */
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
    /* Check if this vertex is an adjacent vertex of the previously
       added vertex. */
    if (graph [ path[pos-1] ][ v ] == 0)
        return false;

    /* Check if the vertex has already been included.
       This step can be optimized by creating an array of size V */
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return false;

    return true;
}

/* A recursive utility function to solve hamiltonian cycle problem */
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
    /* base case: If all vertices are included in Hamiltonian Cycle */
    if (pos == V)
    {
        // And if there is an edge from the last included vertex to the
        // first vertex
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
            return true;
        else
            return false;
    }

    // Try different vertices as a next candidate in Hamiltonian Cycle.
}
```

```

// We don't try for 0 as we included 0 as starting point in hamCycle()
for (int v = 1; v < V; v++)
{
    /* Check if this vertex can be added to Hamiltonian Cycle */
    if (isSafe(v, graph, path, pos))
    {
        path[pos] = v;

        /* recur to construct rest of the path */
        if (hamCycleUtil (graph, path, pos+1) == true)
            return true;

        /* If adding vertex v doesn't lead to a solution,
           then remove it */
        path[pos] = -1;
    }
}

/* If no vertex can be added to Hamiltonian Cycle constructed so far,
   then return false */
return false;
}

/* This function solves the Hamiltonian Cycle problem using Backtracking.
   It mainly uses hamCycleUtil() to solve the problem. It returns false
   if there is no Hamiltonian Cycle possible, otherwise return true and
   prints the path. Please note that there may be more than one solutions,
   this function prints one of the feasible solutions. */
bool hamCycle(bool graph[V][V])
{
    int *path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    /* Let us put vertex 0 as the first vertex in the path. If there is
       a Hamiltonian Cycle, then the path can be started from any point
       of the cycle as the graph is undirected */
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == false )
    {
        printf("\nSolution does not exist");
        return false;
    }

    printSolution(path);
    return true;
}

/* A utility function to print solution */
void printSolution(int path[])
{
    printf ("Solution Exists:\n"
           " Following is one Hamiltonian Cycle \n");
    for (int i = 0; i < V; i++)
        printf(" %d ", path[i]);

    // Let us print the first vertex again to show the complete cycle
    printf(" %d ", path[0]);
    printf("\n");
}

// driver program to test above function
int main()
{
    /* Let us create the following graph
       (0)--(1)--(2)
          |   / \
          |  /
          | /
          (3)-----(4) */
    bool graph1[V][V] = {{0, 1, 0, 1, 0},
                         {1, 0, 1, 1, 1},
                         {0, 1, 0, 0, 1},
                         {1, 1, 0, 0, 1},

```

Run on IDE

Java

```
/* Java program for solution of Hamiltonian Cycle problem
using backtracking */
class HamiltonianCycle
{
    final int V = 5;
    int path[];

    /* A utility function to check if the vertex v can be
       added at index 'pos' in the Hamiltonian Cycle
       constructed so far (stored in 'path[]') */
    boolean isSafe(int v, int graph[][], int path[], int pos)
    {
        /* Check if this vertex is an adjacent vertex of
           the previously added vertex. */
        if (graph[path[pos - 1]][v] == 0)
            return false;

        /* Check if the vertex has already been included.
           This step can be optimized by creating an array
           of size V */
        for (int i = 0; i < pos; i++)
            if (path[i] == v)
                return false;

        return true;
    }

    /* A recursive utility function to solve hamiltonian
       cycle problem */
    boolean hamCycleUtil(int graph[][], int path[], int pos)
    {
        /* base case: If all vertices are included in
           Hamiltonian Cycle */
        if (pos == V)
        {
            // And if there is an edge from the last included
            // vertex to the first vertex
            if (graph[path[pos - 1]][path[0]] == 1)
                return true;
            else
                return false;
        }
    }
}
```

```

// Try different vertices as a next candidate in
// Hamiltonian Cycle. We don't try for 0 as we
// included 0 as starting point in hamCycle()
for (int v = 1; v < V; v++)
{
    /* Check if this vertex can be added to Hamiltonian
       Cycle */
    if (isSafe(v, graph, path, pos))
    {
        path[pos] = v;

        /* recur to construct rest of the path */
        if (hamCycleUtil(graph, path, pos + 1) == true)
            return true;

        /* If adding vertex v doesn't lead to a solution,
           then remove it */
        path[pos] = -1;
    }
}

/* If no vertex can be added to Hamiltonian Cycle
   constructed so far, then return false */
return false;
}

/* This function solves the Hamiltonian Cycle problem using
   Backtracking. It mainly uses hamCycleUtil() to solve the
   problem. It returns false if there is no Hamiltonian Cycle
   possible, otherwise return true and prints the path.
   Please note that there may be more than one solutions,
   this function prints one of the feasible solutions. */
int hamCycle(int graph[][][])
{
    path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    /* Let us put vertex 0 as the first vertex in the path.
       If there is a Hamiltonian Cycle, then the path can be
       started from any point of the cycle as the graph is
       undirected */
    path[0] = 0;
    if (hamCycleUtil(graph, path, 1) == false)
    {
        System.out.println("\nSolution does not exist");
        return 0;
    }

    printSolution(path);
    return 1;
}

/* A utility function to print solution */
void printSolution(int path[])
{
    System.out.println("Solution Exists: Following" +
                       " is one Hamiltonian Cycle");
    for (int i = 0; i < V; i++)
        System.out.print(" " + path[i] + " ");

    // Let us print the first vertex again to show the
    // complete cycle
    System.out.println(" " + path[0] + " ");
}

// driver program to test above function
public static void main(String args[])
{
    HamiltonianCycle hamiltonian =
                    new HamiltonianCycle();
    /* Let us create the following graph
       (0)--(1)--(2)

```

```

        / \
       / \
      / \
     (3)---(4)   */
int graph1[][] = {{0, 1, 0, 1, 0},
                  {1, 0, 1, 1, 1},
                  {0, 1, 0, 0, 1},
                  {1, 1, 0, 0, 1},
                  {0, 1, 1, 1, 0},
                };

// Print the solution
hamiltonian.hamCycle(graph1);

/* Let us create the following graph
   (0)--(1)--(2)
   |   / \   |
   |   /   \  |
   (3)   (4)   */
int graph2[][] = {{0, 1, 0, 1, 0},
                  {1, 0, 1, 1, 1},
                  {0, 1, 0, 0, 1},
                  {1, 1, 0, 0, 0},
                  {0, 1, 1, 0, 0},
                };

// Print the solution
hamiltonian.hamCycle(graph2);
}
// This code is contributed by Abhishek Shankhadhar

```

Run on IDE

Output:

Solution Exists: Following is one Hamiltonian Cycle
 0 1 2 4 3 0

Solution does not exist

Note that the above code always prints cycle starting from 0. Starting point should not matter as cycle can be started from any point. If you want to change the starting point, you should make two changes to above code.

Change "path[0] = 0;" to "path[0] = s;" where s is your new starting point. Also change loop "for (int v = 1; v < V; v++)" in hamCycleUtil() to "for (int v = 0; v < V; v++)". Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Ali Tile BY **LOLA GROUP**

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Graph](#) [Backtracking](#) [Graph](#)

Related Posts:

- Two Clique Problem (Check if Graph can be divided in two Cliques)
- Minimum Product Spanning Tree
- Minimum Cost Path with Left, Right, Bottom and Up moves allowed
- Find if an array of strings can be chained to form a circle | Set 2
- Minimum number of operation required to convert number x into y
- Minimum edges to reverse to make path from a source to a destination
- Maximum edges that can be added to DAG so that it remains DAG
- Minimum steps to reach end of array under constraints

(Login to Rate and Mark)

3.5

Average Difficulty : **3.5/5.0**
Based on **24** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 7 (Sudoku)

Given a partially filled 9×9 2D array ‘grid[9][9]’, the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.

3		6	5		8	4		
5	2							
	8	7				3	1	
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3				2	5		
						7	4	
		5	2		6	3		

Naive Algorithm

The Naive Algorithm is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found.

Backtracking Algorithm

Like all other [Backtracking problems](#), we can solve Sudoku by one by one assigning numbers to empty cells. Before assigning a number, we check whether it is safe to assign. We basically check that the same number is not present in current row, current column and current 3×3 subgrid. After checking for safety, we assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then we try next number for current empty cell. And if none of number (1 to 9) lead to solution, we return false.

```

Find row, col of an unassigned cell
If there is none, return true
For digits from 1 to 9
    a) If there is no conflict for digit at row,col
        assign digit to row,col and recursively try fill in rest of grid
    b) If recursion successful, return true
    c) Else, remove digit and try another
If all digits have been tried and nothing worked, return false
  
```

Following are C++ and Python implementation for Sudoku problem. It prints the completely filled grid as output.

```

// A Backtracking program in C++ to solve Sudoku problem
#include <stdio.h>

// UNASSIGNED is used for empty cells in sudoku grid
#define UNASSIGNED 0

// N is used for size of Sudoku grid. Size will be NxN
#define N 9

// This function finds an entry in grid that is still unassigned
bool FindUnassignedLocation(int grid[N][N], int &row, int &col);

// Checks whether it will be legal to assign num to the given row,col
bool isSafe(int grid[N][N], int row, int col, int num);

/* Takes a partially filled-in grid and attempts to assign values to
   all unassigned locations in such a way to meet the requirements
   for Sudoku solution (non-duplication across rows, columns, and boxes) */
bool SolveSudoku(int grid[N][N])
{
    int row, col;

    // If there is no unassigned location, we are done
    if (!FindUnassignedLocation(grid, row, col))
        return true; // success!

    // consider digits 1 to 9
    for (int num = 1; num <= 9; num++)
    {
        // if looks promising
        if (isSafe(grid, row, col, num))
        {
            // make tentative assignment
            grid[row][col] = num;

            // return, if success, yay!
            if (SolveSudoku(grid))
                return true;

            // failure, unmake & try again
            grid[row][col] = UNASSIGNED;
        }
    }
    return false; // this triggers backtracking
}

/* Searches the grid to find an entry that is still unassigned. If
   found, the reference parameters row, col will be set the location
   that is unassigned, and true is returned. If no unassigned entries
   remain, false is returned. */
bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
{
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}

/* Returns a boolean which indicates whether any assigned entry
   in the specified row matches the given number. */
bool UsedInRow(int grid[N][N], int row, int num)
{
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}

/* Returns a boolean which indicates whether any assigned entry

```

```

        in the specified column matches the given number. */
bool UsedInCol(int grid[N][N], int col, int num)
{
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}

/* Returns a boolean which indicates whether any assigned entry
   within the specified 3x3 box matches the given number. */
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num)
{
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row+boxStartRow][col+boxStartCol] == num)
                return true;
    return false;
}

/* Returns a boolean which indicates whether it will be legal to assign
   num to the given row,col location. */
bool isSafe(int grid[N][N], int row, int col, int num)
{
    /* Check if 'num' is not already placed in current row,
       current column and current 3x3 box */
    return !UsedInRow(grid, row, num) &&
           !UsedInCol(grid, col, num) &&
           !UsedInBox(grid, row - row%3 , col - col%3, num);
}

/* A utility function to print grid */
void printGrid(int grid[N][N])
{
    for (int row = 0; row < N; row++)
    {
        for (int col = 0; col < N; col++)
            printf("%2d", grid[row][col]);
        printf("\n");
    }
}

/* Driver Program to test above functions */
int main()
{
    // 0 means unassigned cells
    int grid[N][N] = {{3, 0, 6, 5, 0, 8, 4, 0, 0},
                      {5, 2, 0, 0, 0, 0, 0, 0, 0},
                      {0, 8, 7, 0, 0, 0, 0, 3, 1},
                      {0, 0, 3, 0, 1, 0, 0, 8, 0},
                      {9, 0, 0, 8, 6, 3, 0, 0, 5},
                      {0, 5, 0, 0, 9, 0, 6, 0, 0},
                      {1, 3, 0, 0, 0, 0, 2, 5, 0},
                      {0, 0, 0, 0, 0, 0, 0, 7, 4},
                      {0, 0, 5, 2, 0, 6, 3, 0, 0}};
    if (SolveSudoku(grid) == true)
        printGrid(grid);
    else
        printf("No solution exists");

    return 0;
}

```

[Run on IDE](#)

Python

```

# A Backtracking program in Python to solve Sudoku problem

# A Utility Function to print the Grid

```

```

def print_grid(arr):
    for i in range(9):
        for j in range(9):
            print arr[i][j],
        print ('\n')

# Function to Find the entry in the Grid that is still not used
# Searches the grid to find an entry that is still unassigned. If
# found, the reference parameters row, col will be set the location
# that is unassigned, and true is returned. If no unassigned entries
# remain, false is returned.
# 'l' is a list variable that has been passed from the solve_sudoku function
# to keep track of incrementation of Rows and Columns
def find_empty_location(arr,l):
    for row in range(9):
        for col in range(9):
            if(arr[row][col]==0):
                l[0]=row
                l[1]=col
                return True
    return False

# Returns a boolean which indicates whether any assigned entry
# in the specified row matches the given number.
def used_in_row(arr,row,num):
    for i in range(9):
        if(arr[row][i] == num):
            return True
    return False

# Returns a boolean which indicates whether any assigned entry
# in the specified column matches the given number.
def used_in_col(arr,col,num):
    for i in range(9):
        if(arr[i][col] == num):
            return True
    return False

# Returns a boolean which indicates whether any assigned entry
# within the specified 3x3 box matches the given number
def used_in_box(arr,row,col,num):
    for i in range(3):
        for j in range(3):
            if(arr[i+row][j+col] == num):
                return True
    return False

# Checks whether it will be legal to assign num to the given row,col
# Returns a boolean which indicates whether it will be legal to assign
# num to the given row,col location.
def check_location_is_safe(arr,row,col,num):

    # Check if 'num' is not already placed in current row,
    # current column and current 3x3 box
    return not used_in_row(arr,row,num) and not used_in_col(arr,col,num) and not used_in_box(arr, row, col, num)

# Takes a partially filled-in grid and attempts to assign values to
# all unassigned locations in such a way to meet the requirements
# for Sudoku solution (non-duplication across rows, columns, and boxes)
def solve_sudoku(arr):

    # 'l' is a list variable that keeps the record of row and col in find_empty_location Function
    l=[0,0]

    # If there is no unassigned location, we are done
    if(not find_empty_location(arr,l)):
        return True

    # Assigning list values to row and col that we got from the above Function
    row=l[0]
    col=l[1]

    # consider digits 1 to 9

```

```

for num in range(1,10):

    # if looks promising
    if(check_location_is_safe(arr,row,col,num)):

        # make tentative assignment
        arr[row][col]=num

        # return, if sucess, ya!
        if(solve_sudoku(arr)):
            return True

        # failure, unmake & try again
        arr[row][col] = 0

    # this triggers backtracking
return False

# Driver main function to test above functions
if __name__=="__main__":
    # creating a 2D array for the grid
    grid=[[0 for x in range(9)]for y in range(9)]

    # assigning values to the grid
    grid=[[3,0,6,5,0,8,4,0,0],
          [5,2,0,0,0,0,0,0,0],
          [0,8,7,0,0,0,0,3,1],
          [0,0,3,0,1,0,0,8,0],
          [9,0,0,8,6,3,0,0,5],
          [0,5,0,0,9,0,6,0,0],
          [1,3,0,0,0,0,2,5,0],
          [0,0,0,0,0,0,0,7,4],
          [0,0,5,2,0,6,3,0,0]]

    # if sucess print the grid
    if(solve_sudoku(grid)):
        print_grid(grid)
    else:
        print "No solution exists"

```

The above code has been contributed by Harshit Sidhwa.

[Run on IDE](#)

Output:

```

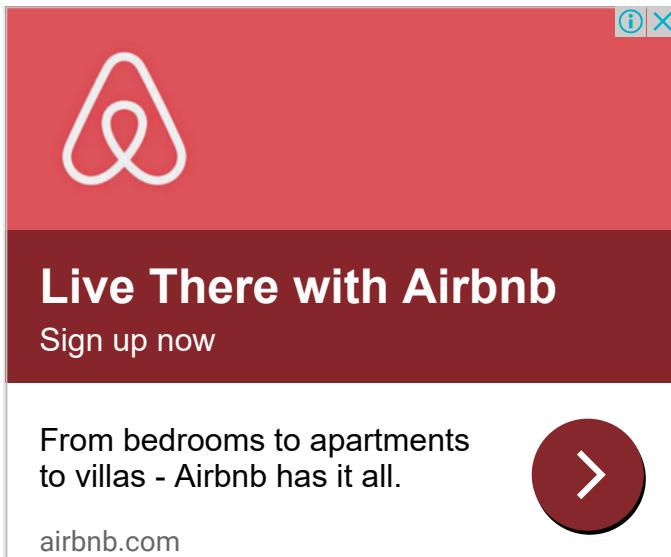
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9

```

References:

<http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



The image shows a vertical advertisement for Airbnb. At the top is a red header with the Airbnb logo. Below it is a dark red section with the text "Live There with Airbnb" and "Sign up now". Underneath is a white section with the text "From bedrooms to apartments to villas - Airbnb has it all." followed by a large orange circular arrow button with a white right-pointing arrow. At the bottom is a white footer with the text "airbnb.com".

GATE CS Corner Company Wise Coding Practice

Backtracking Backtracking

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 28 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Tug of War

Given a set of n integers, divide the set in two subsets of $n/2$ sizes each such that the difference of the sum of two subsets is as minimum as possible. If n is even, then sizes of two subsets must be strictly $n/2$ and if n is odd, then size of one subset must be $(n-1)/2$ and size of other subset must be $(n+1)/2$.

For example, let given set be {3, 4, 5, -3, 100, 1, 89, 54, 23, 20}, the size of set is 10. Output for this set should be {4, 100, 1, 23, 20} and {3, 5, -3, 89, 54}. Both output subsets are of size 5 and sum of elements in both subsets is same (148 and 148).

Let us consider another example where n is odd. Let given set be {23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4}. The output subsets should be {45, -34, 12, 98, -1} and {23, 0, -99, 4, 189, 4}. The sums of elements in two subsets are 120 and 121 respectively.

The following solution tries every possible subset of half size. If one subset of half size is formed, the remaining elements form the other subset. We initialize current set as empty and one by one build it. There are two possibilities for every element, either it is part of current set, or it is part of the remaining elements (other subset). We consider both possibilities for every element. When the size of current set becomes $n/2$, we check whether this solution is better than the best solution available so far. If it is, then we update the best solution.

Following is C++ implementation for Tug of War problem. It prints the required arrays.

```
#include <iostream>
#include <stdlib.h>
#include <limits.h>
using namespace std;

// function that tries every possible solution by calling itself recursively
void TOWUtil(int* arr, int n, bool* curr_elements, int no_of_selected_elements,
             bool* soln, int* min_diff, int sum, int curr_sum, int curr_position)
{
    // checks whether the it is going out of bound
    if (curr_position == n)
        return;

    // checks that the numbers of elements left are not less than the
    // number of elements required to form the solution
    if ((n/2 - no_of_selected_elements) > (n - curr_position))
        return;

    // consider the cases when current element is not included in the solution
    TOWUtil(arr, n, curr_elements, no_of_selected_elements,
            soln, min_diff, sum, curr_sum, curr_position+1);

    // add the current element to the solution
    no_of_selected_elements++;
    curr_sum = curr_sum + arr[curr_position];
    curr_elements[curr_position] = true;

    // checks if a solution is formed
    if (no_of_selected_elements == n/2)
```

```

    {
        // checks if the solution formed is better than the best solution so far
        if (abs(sum/2 - curr_sum) < *min_diff)
        {
            *min_diff = abs(sum/2 - curr_sum);
            for (int i = 0; i<n; i++)
                soln[i] = curr_elements[i];
        }
    }
    else
    {
        // consider the cases where current element is included in the solution
        TOWUtil(arr, n, curr_elements, no_of_selected_elements, soln,
                min_diff, sum, curr_sum, curr_position+1);
    }

    // removes current element before returning to the caller of this function
    curr_elements[curr_position] = false;
}

// main function that generate an arr
void tugOfWar(int *arr, int n)
{
    // the boolean array that contains the inclusion and exclusion of an element
    // in current set. The number excluded automatically form the other set
    bool* curr_elements = new bool[n];

    // The inclusion/exclusion array for final solution
    bool* soln = new bool[n];

    int min_diff = INT_MAX;

    int sum = 0;
    for (int i=0; i<n; i++)
    {
        sum += arr[i];
        curr_elements[i] = soln[i] = false;
    }

    // Find the solution using recursive function TOWUtil()
    TOWUtil(arr, n, curr_elements, 0, soln, &min_diff, sum, 0, 0);

    // Print the solution
    cout << "The first subset is: ";
    for (int i=0; i<n; i++)
    {
        if (soln[i] == true)
            cout << arr[i] << " ";
    }
    cout << "\nThe second subset is: ";
    for (int i=0; i<n; i++)
    {
        if (soln[i] == false)
            cout << arr[i] << " ";
    }
}

// Driver program to test above functions
int main()
{
    int arr[] = {23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    tugOfWar(arr, n);
    return 0;
}

```

Run on IDE

Output:

```

The first subset is: 45 -34 12 98 -1
The second subset is: 23 0 -99 4 189 4

```

This article is compiled by [Ashish Anand](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Live There with Airbnb

From bedrooms to apartments to villas -
Airbnb has it all.

airbnb.com



GATE CS Corner Company Wise Coding Practice

[Backtracking](#) [Backtracking](#)

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

4.5

Average Difficulty : 4.5/5.0
Based on 43 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Backtracking | Set 8 (Solving Cryptarithmetic Puzzles)

Newspapers and magazines often have crypt-arithmetic puzzles of the form:

SEND

+ MORE

MONEY

The goal here is to assign each letter a digit from 0 to 9 so that the arithmetic works out correctly. The rules are that all occurrences of a letter must be assigned the same digit, and no digit can be assigned to more than one letter.

- First, create a list of all the characters that need assigning to pass to Solve
- If all characters are assigned, return true if puzzle is solved, false otherwise
- Otherwise, consider the first unassigned character
- for (every possible choice among the digits not in use)

make that choice and then recursively try to assign the rest of the characters

if recursion sucessful, return true

if !successful, unmake assignment and try another digit

- If all digits have been tried and nothing worked, return false to trigger backtracking

```
/*
 * ExhaustiveSolve
 * -----
 * This is the "not-very-smart" version of cryptarithmetic solver. It takes
 * the puzzle itself (with the 3 strings for the two addends and sum) and a
 * string of letters as yet unassigned. If no more letters to assign
 * then we've hit a base-case, if the current letter-to-digit mapping solves
 * the puzzle, we're done, otherwise we return false to trigger backtracking
 * If we have letters to assign, we take the first letter from that list, and
 * try assigning it the digits from 0 to 9 and then recursively working
 * through solving puzzle from here. If we manage to make a good assignment
 * that works, we've succeeded, else we need to unassign that choice and try
 * another digit. This version is easy to write, since it uses a simple
 * approach (quite similar to permutations if you think about it) but it is
 * not so smart because it doesn't take into account the structure of the
 * puzzle constraints (for example, once the two digits for the addends have
 * been assigned, there is no reason to try anything other than the correct
 * digit for the sum) yet it tries a lot of useless combos regardless
 */
bool ExhaustiveSolve(puzzleT puzzle, string lettersToAssign)
{
```

```

if (lettersToAssign.empty()) // no more choices to make
    return PuzzleSolved(puzzle); // checks arithmetic to see if works
for (int digit = 0; digit <= 9; digit++) // try all digits
{
    if (AssignLetterToDigit(lettersToAssign[0], digit))
    {
        if (ExhaustiveSolve(puzzle, lettersToAssign.substr(1)))
            return true;
        UnassignLetterFromDigit(lettersToAssign[0], digit);
    }
}
return false; // nothing worked, need to backtrack
}

```

Run on IDE

The algorithm above actually has a lot in common with the permutations algorithm, it pretty much just creates all arrangements of the mapping from characters to digits and tries each until one works or all have been successfully tried. For a large puzzle, this could take a while.

A smarter algorithm could take into account the structure of the puzzle and avoid going down dead-end paths. For example, if we assign the characters starting from the ones place and moving to the left, at each stage, we can verify the correctness of what we have so far before we continue onwards. This definitely complicates the code but leads to a tremendous improvement in efficiency, making it much more feasible to solve large puzzles.

Below pseudocode in this case has more special cases, but the same general design

- Start by examining the rightmost digit of the topmost row, with a carry of 0
- If we are beyond the leftmost digit of the puzzle, return true if no carry, false otherwise
- If we are currently trying to assign a char in one of the addends
 - If char already assigned, just recur on row beneath this one, adding value into sum
 - If not assigned, then
 - for (every possible choice among the digits not in use)
 - make that choice and then on row beneath this one, if successful, return true
 - if !successful, unmake assignment and try another digit
 - return false if no assignment worked to trigger backtracking
- Else if trying to assign a char in the sum
 - If char assigned & matches correct,
 - recur on next column to the left with carry, if success return true,
 - If char assigned & doesn't match, return false
 - If char unassigned & correct digit already used, return false
 - If char unassigned & correct digit unused,
 - assign it and recur on next column to left with carry, if success return true
 - return false to trigger backtracking

Source:

<http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf>



GATE CS Corner Company Wise Coding Practice

[Backtracking](#) [Backtracking](#)

Related Posts:

- Partition of a set into K subsets with equal sum
- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps
- Find paths from corner cell to middle cell in maze
- Find if there is a path of more than k length from a source

(Login to Rate and Mark)

4.4

Average Difficulty : 4.4/5.0
Based on 13 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

HANDBOOK OF ALGORITHMS

Section
Branch & Bound

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Branch and Bound | Set 1 (Introduction with 0/1 Knapsack)

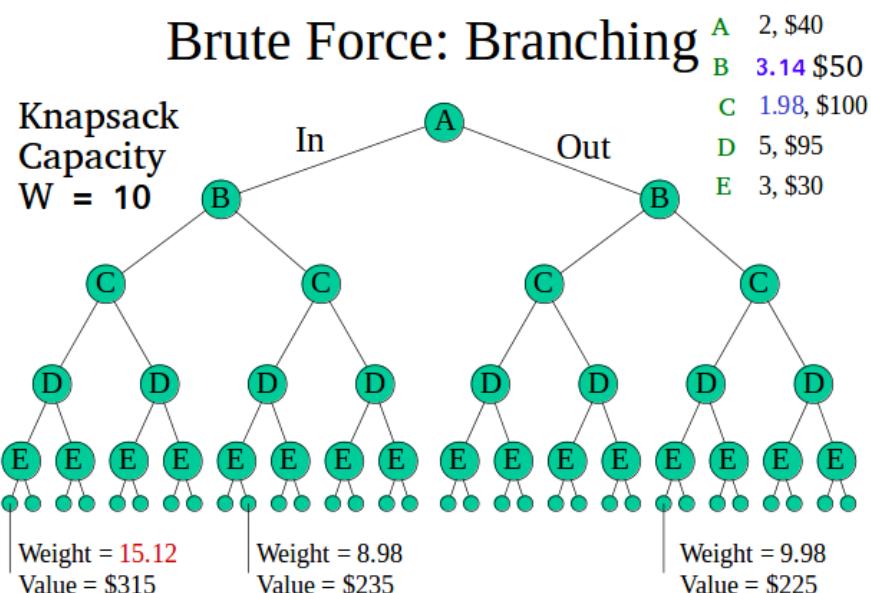
Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. Branch and Bound solve these problems relatively quickly.

Let us consider below 0/1 Knapsack problem to understand Branch and Bound.

Given two integer arrays `val[0..n-1]` and `wt[0..n-1]` that represent values and weights associated with n items respectively. Find out the maximum value subset of `val[]` such that sum of the weights of this subset is smaller than or equal to Knapsack capacity W .

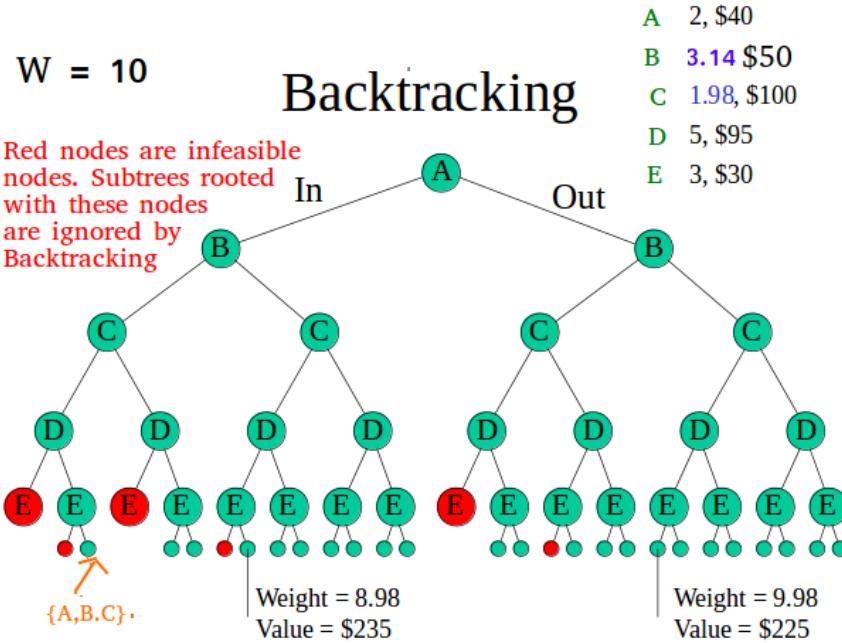
Let us explore all approaches for this problem.

1. A **Greedy approach** is to pick the items in decreasing order of value per unit weight. The Greedy approach works only for **fractional knapsack** problem and may not produce correct result for **0/1 knapsack**.
2. We can use **Dynamic Programming (DP)** for **0/1 Knapsack problem**. In DP, we use a 2D table of size $n \times W$. The **DP Solution doesn't work if item weights are not integers**.
3. Since DP solution doesn't always work, a solution is to use **Brute Force**. With n items, there are 2^n solutions to be generated, check each to see if they satisfy the constraint, save maximum solution that satisfies constraint. This solution can be expressed as **tree**.



4. We can use **Backtracking** to optimize the Brute Force solution. In the tree representation, we can do DFS of tree. If we reach a point where a solution no longer is feasible, there is no need to continue exploring. In the

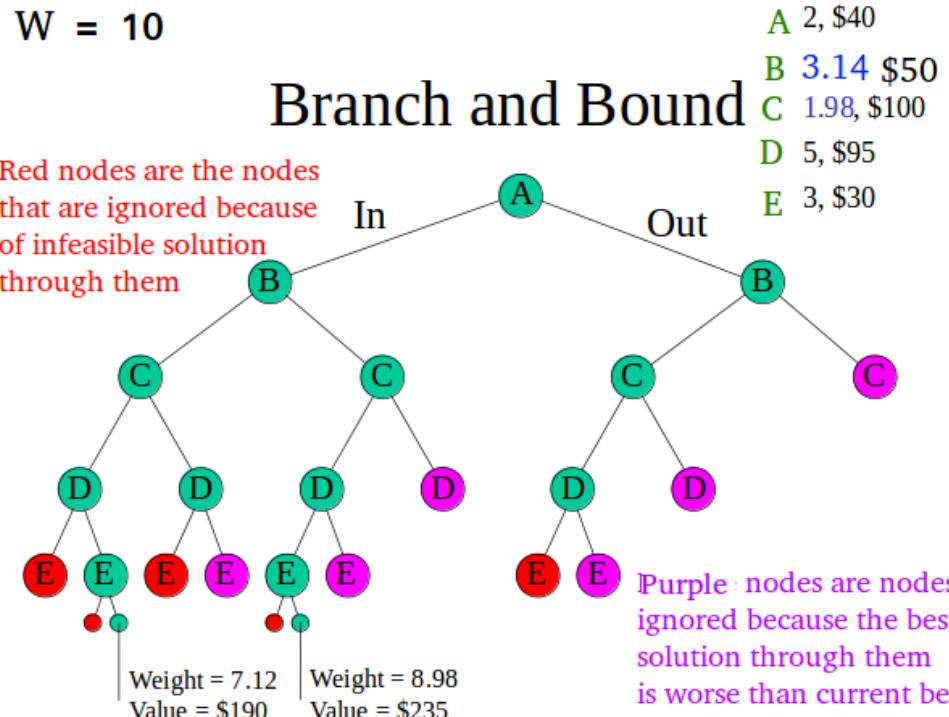
given example, backtracking would be much more effective if we had even more items or a smaller knapsack capacity.



Branch and Bound

The backtracking based solution works better than brute force by ignoring infeasible solutions. We can do better (than backtracking) if we know a bound on best possible solution subtree rooted with every node. If the best in subtree is worse than current best, we can simply ignore this node and its subtrees. So we compute bound (best solution) for every node and compare the bound with current best solution before exploring the node.

Example bounds used in below diagram are, **A** down can give \$315, **B** down can \$275, **C** down can \$225, **D** down can \$125 and **E** down can \$30. In the [next article](#), we have discussed the process to get these bounds.



Branch and bound is very useful technique for searching a solution but in worst case, we need to fully calculate the entire tree. At best, we only need to fully calculate one path through the tree and prune the rest of it.

Source:

Above images and content is adopted from following nice link.

<http://www.cse.msu.edu/~torg/Classes/Archives/cse830.03fall/Lectures/Lecture11.ppt>

Branch and Bound | Set 2 (Implementation of 0/1 Knapsack)

This article is contributed [Utkarsh Trivedi](#). If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Branch and Bound

Related Posts:

- [Branch And Bound | Set 6 \(Traveling Salesman Problem\)](#)
- [Branch And Bound | Set 4 \(Job Assignment Problem\)](#)
- [Branch and Bound | Set 5 \(N Queen Problem\)](#)
- [Branch and Bound | Set 3 \(8 puzzle Problem\)](#)
- [Branch and Bound | Set 2 \(Implementation of 0/1 Knapsack\)](#)

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 4 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Branch and Bound | Set 2 (Implementation of 0/1 Knapsack)

We strongly recommend to refer below post as a prerequisite for this.

[Branch and Bound | Set 1 \(Introduction with 0/1 Knapsack\)](#)

We discussed different approaches to solve above problem and saw that the Branch and Bound solution is the best suited method when item weights are not integers.

In this post implementation of Branch and Bound method for 0/1 knapsack problem is discussed.

How to find bound for every node for 0/1 Knapsack?

The idea is to use the fact that the [Greedy approach](#) provides the best solution for Fractional Knapsack problem.

To check if a particular node can give us a better solution or not, we compute the optimal solution (through the node) using Greedy approach. If the solution computed by Greedy approach itself is more than the best so far, then we can't get a better solution through the node.

Complete Algorithm:

1. Sort all items in decreasing order of ratio of value per unit weight so that an upper bound can be computed using Greedy Approach.
2. Initialize maximum profit, maxProfit = 0
3. Create an empty queue, Q.
4. Create a dummy node of decision tree and enqueue it to Q. Profit and weight of dummy node are 0.
5. Do following while Q is not empty.
 - Extract an item from Q. Let the extracted item be u.
 - Compute profit of next level node. If the profit is more than maxProfit, then update maxProfit.
 - Compute bound of next level node. If bound is more than maxProfit, then add next level node to Q.
 - Consider the case when next level node is not considered as part of solution and add a node to queue with level as next, but weight and profit without considering next level nodes.

Illustration:

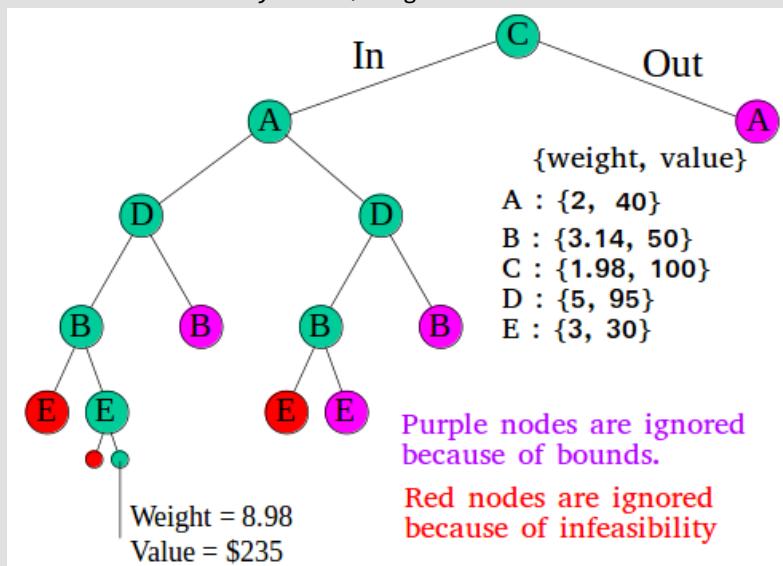
Input:

```
// First thing in every pair is weight of item
// and second thing is value of item
Item arr[] = {{2, 40}, {3.14, 50}, {1.98, 100},
              {5, 95}, {3, 30}};
Knapsack Capacity W = 10
```

Output:

The maximum possible profit = 235

Below diagram shows illustration. Items are considered sorted by value/weight.



Note : The image doesn't strictly follow the algorithm/code as there is no dummy node in the image.

This image is adopted from [here](#).

Following is C++ implementation of above idea.

```
// C++ program to solve knapsack problem using
// branch and bound
#include <bits/stdc++.h>
using namespace std;

// Structure for Item which store weight and corresponding
// value of Item
struct Item
{
    float weight;
    int value;
};

// Node structure to store information of decision
// tree
struct Node
{
    // level --> Level of node in decision tree (or index
    //           in arr[])
    // profit --> Profit of nodes on path from root to this
    //           node (including this node)
    // bound ---> Upper bound of maximum profit in subtree
    //           of this node/
    int level, profit, bound;
    float weight;
};
```

```

// Comparison function to sort Item according to
// val/weight ratio
bool cmp(Item a, Item b)
{
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

// Returns bound of profit in subtree rooted with u.
// This function mainly uses Greedy solution to find
// an upper bound on maximum profit.
int bound(Node u, int n, int W, Item arr[])
{
    // if weight overcomes the knapsack capacity, return
    // 0 as expected bound
    if (u.weight >= W)
        return 0;

    // initialize bound on profit by current profit
    int profit_bound = u.profit;

    // start including items from index 1 more to current
    // item index
    int j = u.level + 1;
    int totweight = u.weight;

    // checking index condition and knapsack capacity
    // condition
    while ((j < n) && (totweight + arr[j].weight <= W))
    {
        totweight += arr[j].weight;
        profit_bound += arr[j].value;
        j++;
    }

    // If k is not n, include last item partially for
    // upper bound on profit
    if (j < n)
        profit_bound += (W - totweight) * arr[j].value /
                        arr[j].weight;
}

return profit_bound;
}

// Returns maximum profit we can get with capacity W
int knapsack(int W, Item arr[], int n)
{
    // sorting Item on basis of value per unit
    // weight.
    sort(arr, arr + n, cmp);

    // make a queue for traversing the node
    queue<Node> Q;
    Node u, v;

    // dummy node at starting
    u.level = -1;
}

```

```

u.profit = u.weight = 0;
Q.push(u);

// One by one extract an item from decision tree
// compute profit of all children of extracted item
// and keep saving maxProfit
int maxProfit = 0;
while (!Q.empty())
{
    // Dequeue a node
    u = Q.front();
    Q.pop();

    // If it is starting node, assign level 0
    if (u.level == -1)
        v.level = 0;

    // If there is nothing on next level
    if (u.level == n-1)
        continue;

    // Else if not last node, then increment level,
    // and compute profit of children nodes.
    v.level = u.level + 1;

    // Taking current level's item add current
    // level's weight and value to node u's
    // weight and value
    v.weight = u.weight + arr[v.level].weight;
    v.profit = u.profit + arr[v.level].value;

    // If cumulated weight is less than W and
    // profit is greater than previous profit,
    // update maxprofit
    if (v.weight <= W && v.profit > maxProfit)
        maxProfit = v.profit;

    // Get the upper bound on profit to decide
    // whether to add v to Q or not.
    v.bound = bound(v, n, W, arr);

    // If bound value is greater than profit,
    // then only push into queue for further
    // consideration
    if (v.bound > maxProfit)
        Q.push(v);

    // Do the same thing, but Without taking
    // the item in knapsack
    v.weight = u.weight;
    v.profit = u.profit;
    v.bound = bound(v, n, W, arr);
    if (v.bound > maxProfit)
        Q.push(v);
}

return maxProfit;
}

```

```
// driver program to test above function
int main()
{
    int W = 10; // Weight of knapsack
    Item arr[] = {{2, 40}, {3.14, 50}, {1.98, 100},
                  {5, 95}, {3, 30}};
    int n = sizeof(arr) / sizeof(arr[0]);

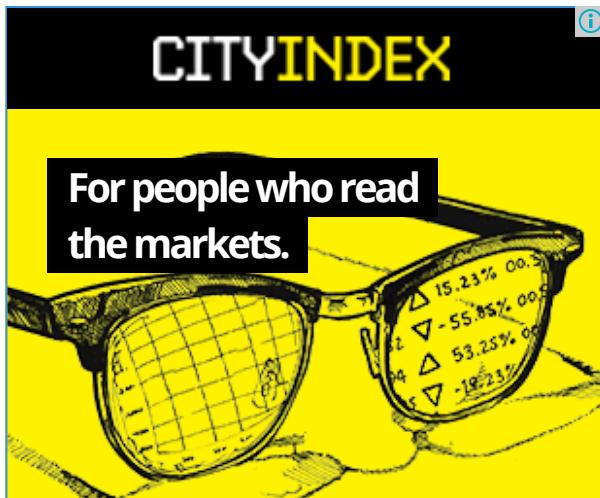
    cout << "Maximum possible profit = "
         << knapsack(W, arr, n);

    return 0;
}
```

Output :

Maximum possible profit = 235

This article is contributed [Utkarsh Trivedi](#). If you likeGeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.



GATE CS Corner Company Wise Coding Practice

[Branch and Bound](#)

Related Posts:

- [Branch And Bound | Set 6 \(Traveling Salesman Problem\)](#)
- [Branch And Bound | Set 4 \(Job Assignment Problem\)](#)
- [Branch and Bound | Set 5 \(N Queen Problem\)](#)
- [Branch and Bound | Set 3 \(8 puzzle Problem\)](#)
- [Branch and Bound | Set 1 \(Introduction with 0/1 Knapsack\)](#)

(Login to Rate and Mark)

0

Average Difficulty : 0/5.0
No votes yet.



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Branch and Bound | Set 3 (8 puzzle Problem)

We have introduced Branch and Bound and discussed 0/1 Knapsack problem in below posts.

- [Branch and Bound | Set 1 \(Introduction with 0/1 Knapsack\)](#)
- [Branch and Bound | Set 2 \(Implementation of 0/1 Knapsack\)](#)

In this puzzle solution of 8 puzzle problem is discussed.

Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

For example,

Initial Configuration

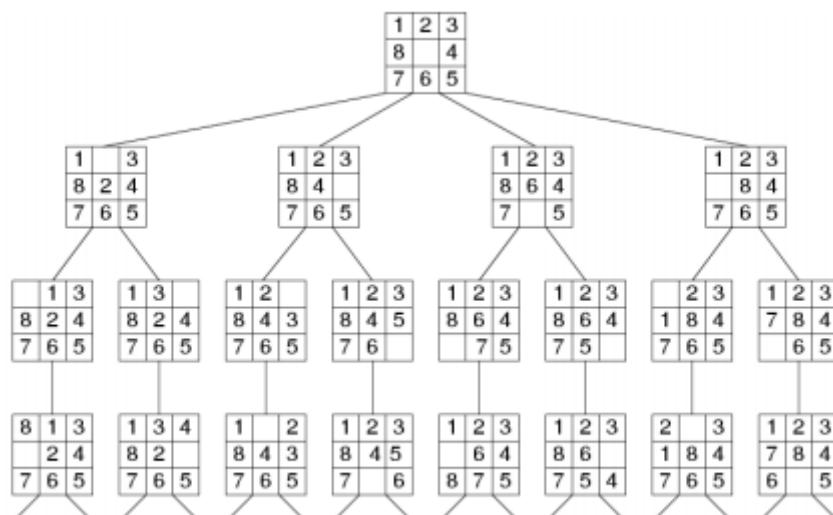
1	2	3
5	6	
7	8	4

Final Configuration

1	2	3
5	8	6
	7	4

1. DFS (Brute-Force)

We can perform depth-first search on state space (Set of all configurations of a given problem i.e. all states that can be reached from initial state) tree.



State Space Tree for 8 Puzzle

Image source: <https://courses.cs.washington.edu/courses/cse473/12au/slides/lect3.pdf>

In this solution, successive moves can take us away from the goal rather than bringing closer. The search of state space tree follows leftmost path from the root regardless of initial state. An answer node may never be found in this approach.

2. BFS (Brute-Force)

We can perform a Breadth-first search on state space tree. This always finds a goal state nearest to the root. But no matter what the initial state is, the algorithm attempts the same sequence of moves like DFS.

3. Branch and Bound

The search for an answer node can often be speeded by using an “intelligent” ranking function, also called an approximate cost function to avoid searching in sub-trees that do not contain an answer node. It is similar to backtracking technique but uses BFS-like search.

There are basically three types of nodes involved in Branch and Bound

1. **Live node** is a node that has been generated but whose children have not yet been generated.
2. **E-node** is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.
3. **Dead node** is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.

Cost function:

Each node X in the search tree is associated with a cost. The cost function is useful for determining the next E-node. The next E-node is the one with least cost. The cost function is defined as,

$C(X) = g(X) + h(X)$ where
 $g(X)$ = cost of reaching the current node
 from the root
 $h(X)$ = cost of reaching an answer node from X.

Ideal Cost function for 8-puzzle Algorithm :

We assume that moving one tile in any direction will have 1 unit cost. Keeping that in mind, we define cost function for 8-puzzle algorithm as below :

$c(x) = f(x) + h(x)$ where
 $f(x)$ is the length of the path from root to x
 (the number of moves so far) and
 $h(x)$ is the number of non-blank tiles not in
 their goal position (the number of mis-
 -placed tiles). There are at least $h(x)$
 moves to transform state x to a goal state

An algorithm is available for getting an approximation of $h(x)$ which is a unknown value.

Complete Algorithm:

```
/* Algorithm LCSearch uses c(x) to find an answer node
 * LCSearch uses Least() and Add() to maintain the list
   of live nodes
```

```

* Least() finds a live node with least c(x), deletes
it from the list and returns it
* Add(x) adds x to the list of live nodes
* Implement list of live nodes as a min heap */

struct list_node
{
    list_node *next;

    // Helps in tracing path when answer is found
    list_node *parent;
    float cost;
}

algorithm LCSearch(list_node *t)
{
    // Search t for an answer node
    // Input: Root node of tree t
    // Output: Path from answer node to root
    if (*t is an answer node)
    {
        print(*t);
        return;
    }

    E = t; // E-node

    Initialize the list of live nodes to be empty;
    while (true)
    {
        for each child x of E
        {
            if x is an answer node
            {
                print the path from x to t;
                return;
            }
            Add (x); // Add x to list of live nodes;
            x->parent = E; // Pointer for path to root
        }

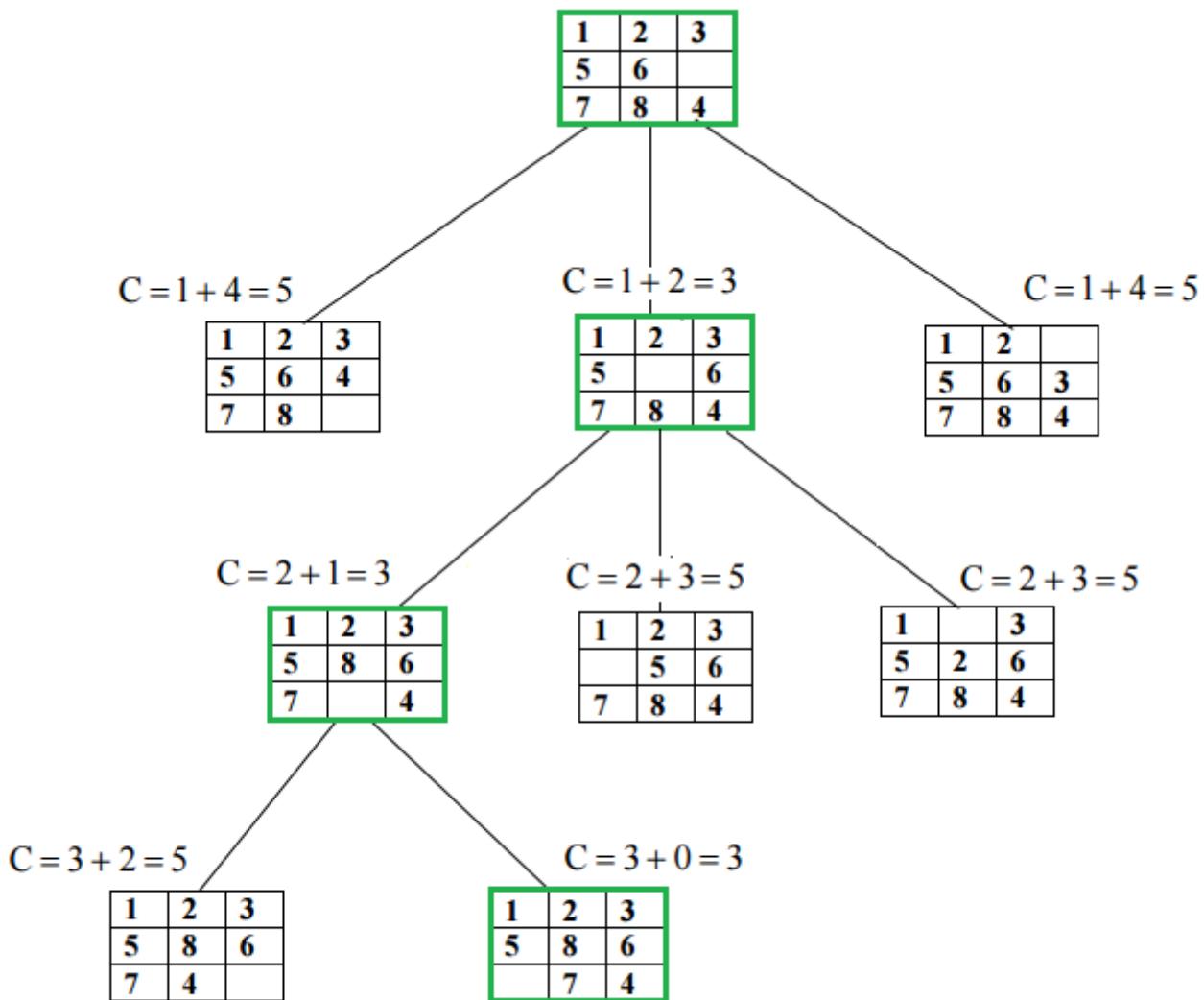
        if there are no more live nodes
        {
            print ("No answer node");
            return;
        }

        // Find a live node with least estimated cost
        E = Least();

        // The found node is deleted from the list of
        // live nodes
    }
}

```

Below diagram shows the path followed by above algorithm to reach final configuration from given initial configuration of 8-Puzzle. Note that only nodes having least value of cost function are expanded.



```
// Program to print path from root node to destination node
// for N*N -1 puzzle algorithm using Branch and Bound
// The solution assumes that instance of puzzle is solvable
#include <bits/stdc++.h>
using namespace std;
#define N 3

// state space tree nodes
struct Node
{
    // stores parent node of current node
    // helps in tracing path when answer is found
    Node* parent;

    // stores matrix
    int mat[N][N];

    // stores blank tile coordinates
    int x, y;

    // stores the number of misplaced tiles
    int cost;

    // stores the number of moves so far
    int level;
};

// Function to print N x N matrix
int printMatrix(int mat[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
}
```

```

        for (int j = 0; j < N; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }

// Function to allocate a new node
Node* newNode(int mat[N][N], int x, int y, int newX,
              int newY, int level, Node* parent)
{
    Node* node = new Node;

    // set pointer for path to root
    node->parent = parent;

    // copy data from parent node to current node
    memcpy(node->mat, mat, sizeof(node->mat));

    // move tile by 1 position
    swap(node->mat[x][y], node->mat[newX][newY]);

    // set number of misplaced tiles
    node->cost = INT_MAX;

    // set number of moves so far
    node->level = level;

    // update new blank tile coordinates
    node->x = newX;
    node->y = newY;

    return node;
}

// bottom, left, top, right
int row[] = { 1, 0, -1, 0 };
int col[] = { 0, -1, 0, 1 };

// Function to calculate the the number of misplaced tiles
// ie. number of non-blank tiles not in their goal position
int calculateCost(int initial[N][N], int final[N][N])
{
    int count = 0;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (initial[i][j] && initial[i][j] != final[i][j])
                count++;
    return count;
}

// Function to check if (x, y) is a valid matrix coordinate
int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}

// print path from root node to destination node
void printPath(Node* root)
{
    if (root == NULL)
        return;
    printPath(root->parent);
    printMatrix(root->mat);

    printf("\n");
}

// Comparison object to be used to order the heap
struct comp
{
    bool operator()(const Node* lhs, const Node* rhs) const
    {
        return (lhs->cost + lhs->level) > (rhs->cost + rhs->level);
    }
}

```

```

};

// Function to solve N*N - 1 puzzle algorithm using
// Branch and Bound. x and y are blank tile coordinates
// in initial state
void solve(int initial[N][N], int x, int y,
           int final[N][N])
{
    // Create a priority queue to store live nodes of
    // search tree;
    priority_queue<Node*, std::vector<Node*>, comp> pq;

    // create a root node and calculate its cost
    Node* root = newNode(initial, x, y, x, y, 0, NULL);
    root->cost = calculateCost(initial, final);

    // Add root to list of live nodes;
    pq.push(root);

    // Finds a live node with least cost,
    // add its childrens to list of live nodes and
    // finally deletes it from the list.
    while (!pq.empty())
    {
        // Find a live node with least estimated cost
        Node* min = pq.top();

        // The found node is deleted from the list of
        // live nodes
        pq.pop();

        // if min is an answer node
        if (min->cost == 0)
        {
            // print the path from root to destination;
            printPath(min);
            return;
        }

        // do for each child of min
        // max 4 children for a node
        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y + col[i]))
            {
                // create a child node and calculate
                // its cost
                Node* child = newNode(min->mat, min->x,
                                      min->y, min->x + row[i],
                                      min->y + col[i],
                                      min->level + 1, min);
                child->cost = calculateCost(child->mat, final);

                // Add child to list of live nodes
                pq.push(child);
            }
        }
    }
}

// Driver code
int main()
{
    // Initial configuration
    // Value 0 is used for empty space
    int initial[N][N] =
    {
        {1, 2, 3},
        {5, 6, 0},
        {7, 8, 4}
    };

    // Solvable Final configuration
    // Value 0 is used for empty space
}

```

```

int final[N][N] =
{
    {1, 2, 3},
    {5, 8, 6},
    {0, 7, 4}
};

// Blank tile coordinates in initial
// configuration
int x = 1, y = 2;

solve(initial, x, y, final);

return 0;
}

```

[Run on IDE](#)

Output :

```

1 2 3
5 6 0
7 8 4

```

```

1 2 3
5 0 6
7 8 4

```

```

1 2 3
5 8 6
7 0 4

```

```

1 2 3
5 8 6
0 7 4

```

Sources:

www.cs.umsl.edu/~sanjiv/classes/cs5130/lectures/bb.pdf
https://www.seas.gwu.edu/~bell/csci212/Branch_and_Bound.pdf

This article is contributed by **Aditya Goel**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Ali Tile BY LOLA GROUP

LOLA CERAMICS (M) SDN. BHD.
ADDRESS: 38, Jalan P-20/345, Taman Durianbaru, 4719 Petaling Jaya, Malaysia
WEBSITE: www.alatile.com

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alatile.com

GATE CS Corner Company Wise Coding Practice

Branch and Bound

Related Posts:

- Branch And Bound | Set 6 (Traveling Salesman Problem)
- Branch And Bound | Set 4 (Job Assignment Problem)
- Branch and Bound | Set 5 (N Queen Problem)
- Branch and Bound | Set 2 (Implementation of 0/1 Knapsack)
- Branch and Bound | Set 1 (Introduction with 0/1 Knapsack)

(Login to Rate and Mark)

0

Average Difficulty : 0/5.0
No votes yet.



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Branch And Bound | Set 4 (Job Assignment Problem)

Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Worker A takes 8 units of time to finish job 4.

An example job assignment problem. Green values show optimal job assignment that is A-Job4, B-Job1 C-Job3 and D-Job4

Let us explore all approaches for this problem.

Solution 1: Brute Force

We generate $n!$ possible job assignments and for each such assignment, we compute its total cost and return the less expensive assignment. Since the solution is a permutation of the n jobs, its complexity is $O(n!)$.

Solution 2: Hungarian Algorithm

The optimal assignment can be found using the Hungarian algorithm. The Hungarian algorithm has worst case run-time complexity of $O(n^3)$.

Solution 3: DFS/BFS on state space tree

A state space tree is a N -ary tree with property that any path from root to leaf node holds one of many solutions to given problem. We can perform depth-first search on state space tree and but successive moves can take us away from the goal rather than bringing closer. The search of state space tree follows leftmost path from the root regardless of initial state. An answer node may never be found in this approach. We can also perform a Breadth-first search on state space tree. But no matter what the initial state is, the algorithm attempts the same sequence of moves like DFS.

Solution 4: Finding Optimal Solution using Branch and Bound

The selection rule for the next node in BFS and DFS is “blind”. i.e. the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly. The search for an optimal solution can often be speeded by using an “intelligent” ranking function, also called an approximate cost function to

avoid searching in sub-trees that do not contain an optimal solution. It is similar to BFS-like search but with one major optimization. Instead of following FIFO order, we choose a live node with least cost. We may not get optimal solution by following node with least promising cost, but it will provide very good chance of getting the search to an answer node quickly.

There are two approaches to calculate the cost function:

1. For each worker, we choose job with minimum cost from list of unassigned jobs (take minimum entry from each row).
2. For each job, we choose a worker with lowest cost for that job from list of unassigned workers (take minimum entry from each column).

In this article, the first approach is followed.

Let's take below example and try to calculate promising cost when Job 2 is assigned to worker A.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Since Job 2 is assigned to worker A (marked in green), cost becomes 2 and Job 2 and worker A becomes unavailable (marked in red).

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Now we assign job 3 to worker B as it has minimum cost from list of unassigned jobs. Cost becomes $2 + 3 = 5$ and Job 3 and worker B also becomes unavailable.

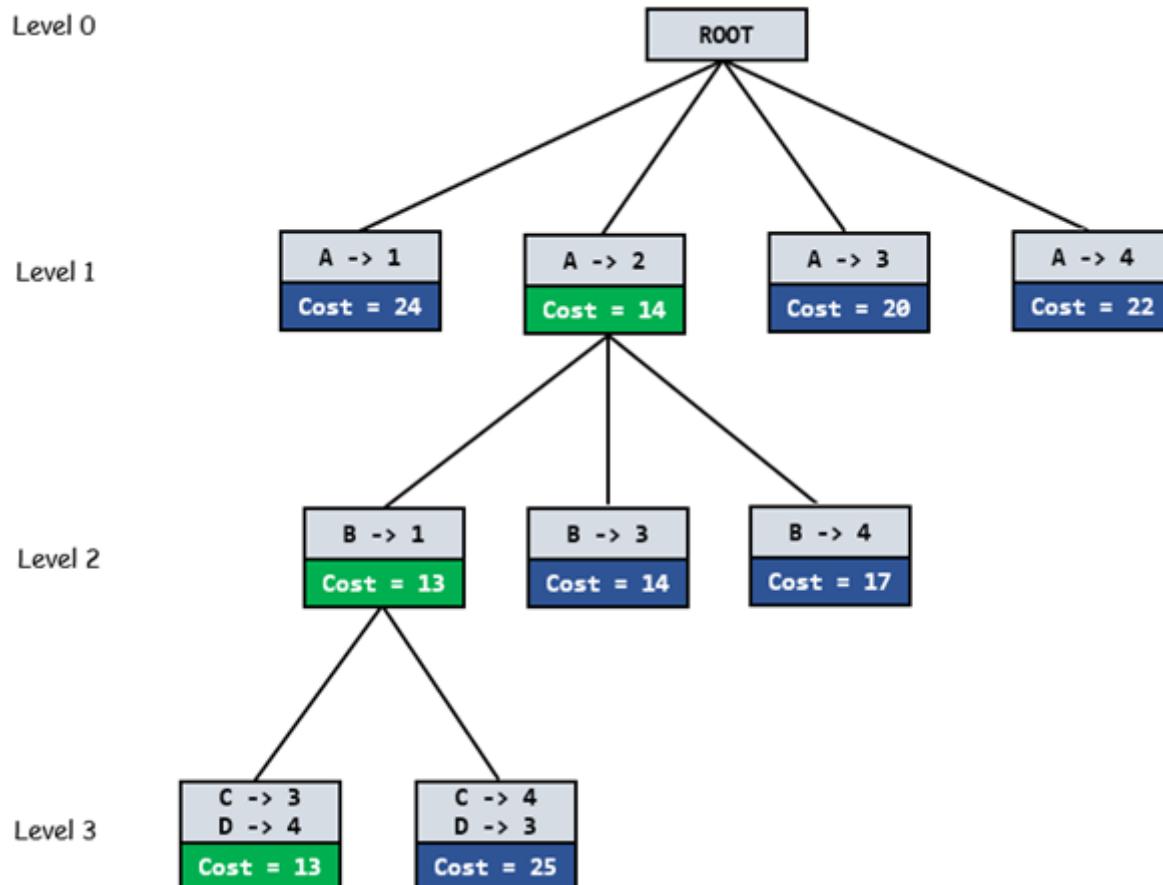
	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Finally, job 1 gets assigned to worker C as it has minimum cost among unassigned jobs and job 4 gets assigned to

worker C as it is only Job left. Total cost becomes $2 + 3 + 5 + 4 = 14$.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Below diagram shows complete search space diagram showing optimal solution path in green.



Complete Algorithm:

```

/* findMinCost uses Least() and Add() to maintain the
list of live nodes

Least() finds a live node with least cost, deletes
it from the list and returns it

Add(x) calculates cost of x and adds it to the list
of live nodes

Implements list of live nodes as a min heap */
  
```

```

// Search Space Tree Node
node
{
  
```

```

int job_number;
int worker_number;
node parent;
int cost;
}

// Input: Cost Matrix of Job Assignment problem
// Output: Optimal cost and Assignment of Jobs
algorithm findMinCost (costMatrix mat[][])
{
    // Initialize list of live nodes(min-Heap)
    // with root of search tree i.e. a Dummy node
    while (true)
    {
        // Find a live node with least estimated cost
        E = Least();

        // The found node is deleted from the list
        // of live nodes
        if (E is a leaf node)
        {
            printSolution();
            return;
        }

        for each child x of E
        {
            Add(x); // Add x to list of live nodes;
            x->parent = E; // Pointer for path to root
        }
    }
}

```

Below is its C++ implementation.

```

// Program to solve Job Assignment problem
// using Branch and Bound
#include <bits/stdc++.h>
using namespace std;
#define N 4

// state space tree node
struct Node
{
    // stores parent node of current node
    // helps in tracing path when answer is found
    Node* parent;

    // contains cost for ancestors nodes
    // including current node
    int pathCost;

    // contains least promising cost
    int cost;

    // contain worker number
    int workerID;

    // contains Job ID
    int jobID;

    // Boolean array assigned will contains
    // info about available jobs
    bool assigned[N];
};

```

```

};

// Function to allocate a new search tree node
// Here Person x is assigned to job y
Node* newNode(int x, int y, bool assigned[],
              Node* parent)
{
    Node* node = new Node;

    for (int j = 0; j < N; j++)
        node->assigned[j] = assigned[j];
    node->assigned[y] = true;

    node->parent = parent;
    node->workerID = x;
    node->jobID = y;

    return node;
}

// Function to calculate the least promising cost
// of node after worker x is assigned to job y.
int calculateCost(int costMatrix[N][N], int x,
                  int y, bool assigned[])
{
    int cost = 0;

    // to store unavailable jobs
    bool available[N] = {true};

    // start from next worker
    for (int i = x + 1; i < N; i++)
    {
        int min = INT_MAX, minIndex = -1;

        // do for each job
        for (int j = 0; j < N; j++)
        {
            // if job is unassigned
            if (!assigned[j] && available[j] &&
                costMatrix[i][j] < min)
            {
                // store job number
                minIndex = j;

                // store cost
                min = costMatrix[i][j];
            }
        }

        // add cost of next worker
        cost += min;

        // job becomes unavailable
        available[minIndex] = false;
    }

    return cost;
}

// Comparison object to be used to order the heap
struct comp
{
    bool operator()(const Node* lhs,
                    const Node* rhs) const
    {
        return lhs->cost > rhs->cost;
    }
};

// print Assignments
void printAssignments(Node *min)
{
    if(min->parent==NULL)

```

```

    return;

    printAssignments(min->parent);
    cout << "Assign Worker " << char(min->workerID + 'A')
    << " to Job " << min->jobID << endl;
}

// Finds minimum cost using Branch and Bound.
int findMinCost(int costMatrix[N][N])
{
    // Create a priority queue to store live nodes of
    // search tree;
    priority_queue<Node*, std::vector<Node*>, comp> pq;

    // initailize heap to dummy node with cost 0
    bool assigned[N] = {false};
    Node* root = newNode(-1, -1, assigned, NULL);
    root->pathCost = root->cost = 0;
    root->workerID = -1;

    // Add dummy node to list of live nodes;
    pq.push(root);

    // Finds a live node with least cost,
    // add its childrens to list of live nodes and
    // finally deletes it from the list.
    while (!pq.empty())
    {
        // Find a live node with least estimated cost
        Node* min = pq.top();

        // The found node is deleted from the list of
        // live nodes
        pq.pop();

        // i stores next worker
        int i = min->workerID + 1;

        // if all workers are assigned a job
        if (i == N)
        {
            printAssignments(min);
            return min->cost;
        }

        // do for each job
        for (int j = 0; j < N; j++)
        {
            // If unassigned
            if (!min->assigned[j])
            {
                // create a new tree node
                Node* child = newNode(i, j, min->assigned, min);

                // cost for ancestors nodes including current node
                child->pathCost = min->pathCost + costMatrix[i][j];

                // calculate its lower bound
                child->cost = child->pathCost +
                    calculateCost(costMatrix, i, j, child->assigned);

                // Add child to list of live nodes;
                pq.push(child);
            }
        }
    }

    // Driver code
    int main()
    {
        // x-coordinate represents a Worker
        // y-coordinate represents a Job
    }
}

```

```

int costMatrix[N][N] =
{
    {9, 2, 7, 8},
    {6, 4, 3, 7},
    {5, 8, 1, 8},
    {7, 6, 9, 4}
};

/* int costMatrix[N][N] =
{
    {82, 83, 69, 92},
    {77, 37, 49, 92},
    {11, 69, 5, 86},
    {8, 9, 98, 23}
};

/* int costMatrix[N][N] =
{
    {2500, 4000, 3500},
    {4000, 6000, 3500},
    {2000, 4000, 2500}
}; */

/*int costMatrix[N][N] =
{
    {90, 75, 75, 80},
    {30, 85, 55, 65},
    {125, 95, 90, 105},
    {45, 110, 95, 115}
}; */
}

cout << "\nOptimal Cost is "
     << findMinCost(costMatrix);

return 0;
}

```

Run on IDE

Output :

```

Assign Worker A to Job 1
Assign Worker B to Job 0
Assign Worker C to Job 2
Assign Worker D to Job 3

```

```
Optimal Cost is 13
```

Reference :

www.cs.umsl.edu/~sanjiv/classes/cs5130/lectures/bb.pdf

This article is contributed by **Aditya Goel**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Branch and Bound

Related Posts:

- Branch And Bound | Set 6 (Traveling Salesman Problem)
- Branch and Bound | Set 5 (N Queen Problem)
- Branch and Bound | Set 3 (8 puzzle Problem)
- Branch and Bound | Set 2 (Implementation of 0/1 Knapsack)
- Branch and Bound | Set 1 (Introduction with 0/1 Knapsack)

(Login to Rate and Mark)

4.5

Average Difficulty : 4.5/5.0
Based on 4 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

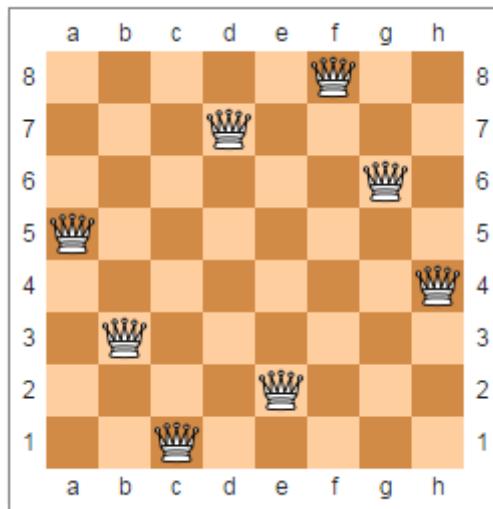


[Login/Register](#)

Branch and Bound | Set 5 (N Queen Problem)

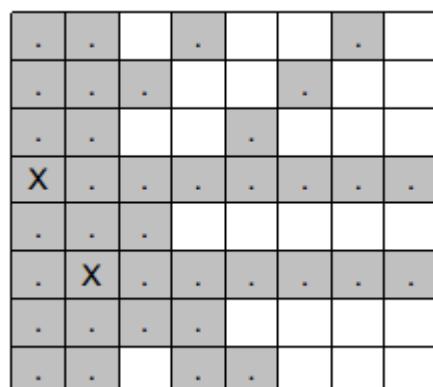
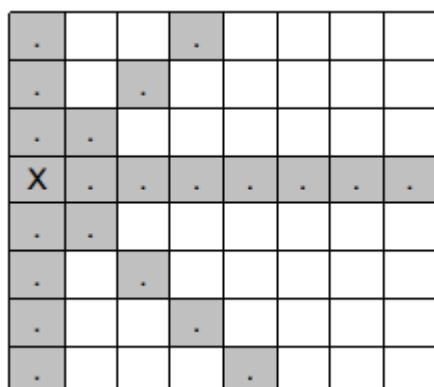
The **N queens puzzle** is the problem of placing N **chess queens** on an $N \times N$ chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

For example, below is one of the solution for famous 8 Queen problem.



Backtracking Algorithm for N-Queen is already discussed [here](#). In backtracking solution we backtrack when we hit a dead end. **In Branch and Bound solution, after building a partial solution, we figure out that there is no point going any deeper as we are going to hit a dead end.**

Let's begin by describing backtracking solution. "The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false."



1. For the 1st Queen, there are total 8 possibilities as we can place 1st Queen in any row of first column. Let's place Queen 1 on row 3.
2. After placing 1st Queen, there are 7 possibilities left for the 2nd Queen. But wait, we don't really have 7 possibilities. We cannot place Queen 2 on rows 2, 3 or 4 as those cells are under attack from Queen 1. So, Queen 2 has only $8 - 3 = 5$ valid positions left.
3. After picking a position for Queen 2, Queen 3 has even fewer options as most of the cells in its column are under attack from the first 2 Queens.

We need to figure out an efficient way of keeping track of which cells are under attack. In previous solution we kept an 8-by-8 Boolean matrix and update it each time we placed a queen, but that required linear time to update as we need to check for safe cells.

Basically, we have to ensure 4 things:

1. No two queens share a column.
2. No two queens share a row.
3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

Number 1 is automatic because of the way we store the solution. For number 2, 3 and 4, we can perform updates in O(1) time. The idea is to keep **three Boolean arrays that tell us which rows and which diagonals are occupied**.

Lets do some pre-processing first. Let's create two $N \times N$ matrix one for / diagonal and other one for \ diagonal. Let's call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /diagonal will have the same value in matrix slashCode, and if they share same \diagonal, they will have the same value in backslashCode matrix.

For an $N \times N$ matrix, fill slashCode and backslashCode matrix using below formula –

$$\text{slashCode}[row][col] = \text{row} + \text{col}$$

$$\text{backslashCode}[row][col] = \text{row} - \text{col} + (N-1)$$

Using above formula will result in below matrices

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

$r+c$

7	6	5	4	3	2	1	0
8	7	6	5	4	3	2	1
9	8	7	6	5	4	3	2
10	9	8	7	6	5	4	3
11	10	9	8	7	6	5	4
12	11	10	9	8	7	6	5
13	12	11	10	9	8	7	6
14	13	12	11	10	9	8	7

$r - c + 7$

The ' $N - 1$ ' in the backslash code is there to ensure that the codes are never negative because we will be using the codes as indices in an array.

Now before we place queen i on row j , we first check whether row j is used (use an array to store row info). Then we check whether slash code ($j + i$) or backslash code ($j - i + 7$) are used (keep two arrays that will tell us which

diagonals are occupied). If yes, then we have to try a different location for queen i. If not, then we mark the row and the two diagonals as used and recurse on queen i + 1. After the recursive call returns and before we try another position for queen i, we need to reset the row, slash code and backslash code as unused again, like in the code from the previous notes.

Below is C++ implementation of above idea –

```

/* C++ program to solve N Queen Problem using Branch
   and Bound */
#include<stdio.h>
#include<string.h>
#define N 8

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%2d ", board[i][j]);
        printf("\n");
    }
}

/* A Optimized function to check if a queen can
be placed on board[row][col] */
bool isSafe(int row, int col, int slashCode[N][N],
            int backslashCode[N][N], bool rowLookup[],
            bool slashCodeLookup[], bool backslashCodeLookup[] )
{
    if (slashCodeLookup[slashCode[row][col]] ||
        backslashCodeLookup[backslashCode[row][col]] ||
        rowLookup[row])
        return false;

    return true;
}

/* A recursive utility function to solve N Queen problem */
bool solveNQueensUtil(int board[N][N], int col,
                      int slashCode[N][N], int backslashCode[N][N], bool rowLookup[N],
                      bool slashCodeLookup[], bool backslashCodeLookup[] )
{
    /* base case: If all queens are placed
       then return true */
    if (col >= N)
        return true;

    /* Consider this column and try placing
       this queen in all rows one by one */
    for (int i = 0; i < N; i++)
    {
        /* Check if queen can be placed on
           board[i][col] */
        if (isSafe(i, col, slashCode, backslashCode, rowLookup,
                  slashCodeLookup, backslashCodeLookup) )
        {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;
            rowLookup[i] = true;
            slashCodeLookup[slashCode[i][col]] = true;
            backslashCodeLookup[backslashCode[i][col]] = true;

            /* recur to place rest of the queens */
            if (solveNQueensUtil(board, col + 1, slashCode, backslashCode,
                                 rowLookup, slashCodeLookup, backslashCodeLookup) )
                return true;

            /* If placing queen in board[i][col]
               doesn't lead to a solution, then backtrack */

            /* Remove queen from board[i][col] */
            board[i][col] = 0;
            rowLookup[i] = false;
            slashCodeLookup[slashCode[i][col]] = false;
            backslashCodeLookup[backslashCode[i][col]] = false;
        }
    }
}

```

```

        board[i][col] = 0;
        rowLookup[i] = false;
        slashCodeLookup[slashCode[i][col]] = false;
        backslashCodeLookup[backslashCode[i][col]] = false;
    }

}

/* If queen can not be place in any row in
   this colum col then return false */
return false;
}

/* This function solves the N Queen problem using
Branch and Bound. It mainly uses solveNQueensUtil() to
solve the problem. It returns false if queens
cannot be placed, otherwise return true and
prints placement of queens in the form of 1s.
Please note that there may be more than one
solutions, this function prints one of the
feasible solutions.*/
bool solveNQueens()
{
    int board[N][N];
    memset(board, 0, sizeof board);

    // helper matrices
    int slashCode[N][N];
    int backslashCode[N][N];

    // arrays to tell us which rows are occupied
    bool rowLookup[N] = {false};

    //keep two arrays to tell us which diagonals are occupied
    bool slashCodeLookup[2*N - 1] = {false};
    bool backslashCodeLookup[2*N - 1] = {false};

    // initialize helper matrices
    for (int r = 0; r < N; r++)
        for (int c = 0; c < N; c++)
            slashCode[r] = r + c,
            backslashCode[r] = r - c + 7;

    if (solveNQueensUtil(board, 0, slashCode, backslashCode,
        rowLookup, slashCodeLookup, backslashCodeLookup) == false )
    {
        printf("Solution does not exist");
        return false;
    }

    // solution found
    printSolution(board);
    return true;
}

// driver program to test above function
int main()
{
    solveNQueens();

    return 0;
}

```

Run on IDE

Output :

```

1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0

```

```

0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

```

Performance:

When run on local machines for N = 32, the backtracking solution took 659.68 seconds while above branch and bound solution took only 119.63 seconds. The difference will be even huge for larger values of N.

Backtracking Algorithm

```

Process returned 0 <0x0>    execution time : 659.686 s
Press any key to continue.

```

Branch and Bound Algorithm

```

Process returned 0 <0x0>    execution time : 119.631 s
Press any key to continue.

```

References :

https://en.wikipedia.org/wiki/Eight_queens_puzzle

www.cs.cornell.edu/~wdtseng/icpc/notes/bt2.pdf

This article is contributed by **Aditya Goel**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

**GATE CS Corner Company Wise Coding Practice**

Branch and Bound

Related Posts:

- Branch And Bound | Set 6 (Traveling Salesman Problem)
- Branch And Bound | Set 4 (Job Assignment Problem)
- Branch and Bound | Set 3 (8 puzzle Problem)
- Branch and Bound | Set 2 (Implementation of 0/1 Knapsack)
- Branch and Bound | Set 1 (Introduction with 0/1 Knapsack)

(Login to Rate and Mark)

0

Average Difficulty : 0/5.0
No votes yet.



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

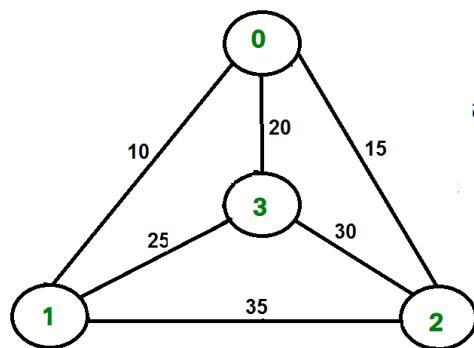
Branch And Bound | Set 6 (Traveling Salesman Problem)

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.

For example, consider the graph shown in figure on right side. A TSP tour in the graph is 0-1-3-2-0. The cost of the tour is $10+25+30+15$ which is 80.

We have discussed following solutions

- 1) [Naive and Dynamic Programming](#)
- 2) [Approximate solution using MST](#)



Branch and Bound Solution

As seen in the previous articles, in Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the subtree rooted with the node.

Note that the cost through a node includes two costs.

- 1) Cost of reaching the node from the root (When we reach a node, we have this cost computed)
- 2) Cost of reaching an answer from current node to a leaf (We compute a bound on this cost to decide whether to ignore subtree with this node or not).

- In cases of a **maximization problem**, an upper bound tells us the maximum possible solution if we follow the given node. For example in [0/1 knapsack](#) we used [Greedy approach to find an upper bound](#).
- In cases of a **minimization problem**, a lower bound tells us the minimum possible solution if we follow the given node. For example, in [Job Assignment Problem](#), we get a lower bound by assigning least cost job to a worker.

In branch and bound, the challenging part is figuring out a way to compute a bound on best possible solution. Below is an idea used to compute bounds for Traveling salesman problem.

Cost of any tour can be written as below.

Cost of a tour $T = (1/2) * \Sigma (\text{Sum of cost of two edges adjacent to } u \text{ and in the})$

tour T)

where $u \in V$

For every vertex u , if we consider two edges through it in T , and sum their costs. The overall sum for all vertices would be twice of cost of tour T (We have considered every edge twice.)

(Sum of two tour edges adjacent to u) \geq (sum of minimum weight
two edges adjacent to
 u)

Cost of any tour $\geq 1/2 * \sum (\text{Sum of cost of two minimum weight edges adjacent to } u)$
where $u \in V$

For example, consider the above shown graph. Below are minimum cost two edges adjacent to every node.

Node	Least cost edges	Total cost
0	(0, 1), (0, 2)	25
1	(0, 1), (1, 3)	35
2	(0, 2), (2, 3)	45
3	(0, 3), (1, 3)	45

Thus a lower bound on the cost of any tour =

$$1/2(25 + 35 + 45 + 45)$$

 $= 75$

Refer [this](#) for one more example.

Now we have an idea about computation of lower bound. Let us see how to apply it state space search tree. We start enumerating all possible nodes (preferably in lexicographical order)

1. The Root Node: Without loss of generality, we assume we start at vertex "0" for which the lower bound has been calculated above.

Dealing with Level 2: The next level enumerates all possible vertices we can go to (keeping in mind that in any path a vertex has to occur only once) which are, 1, 2, 3... n (Note that the graph is complete). Consider we are calculating for vertex 1, Since we moved from 0 to 1, our tour has now included the edge 0-1. This allows us to make necessary changes in the lower bound of the root.

Lower Bound for vertex 1 =
Old lower bound - ((minimum edge cost of 0 +
minimum edge cost of 1) / 2)
+ (edge cost 0-1)

How does it work? To include edge 0-1, we add the edge cost of 0-1, and subtract an edge weight such that the lower bound remains as tight as possible which would be the sum of the minimum edges of 0 and 1 divided by 2. Clearly, the edge subtracted can't be smaller than this.

Dealing with other levels: As we move on to the next level, we again enumerate all possible vertices. For the above case going further after 1, we check out for 2, 3, 4, ...n.

Consider lower bound for 2 as we moved from 1 to 1, we include the edge 1-2 to the tour and alter the new lower bound for this node.

```

Lower bound(2) =
    Old lower bound - ((second minimum edge cost of 1 +
                        minimum edge cost of 2)/2)
                        + edge cost 1-2)

```

Note: The only change in the formula is that this time we have included second minimum edge cost for 1, because the minimum edge cost has already been subtracted in previous level.

```

// C++ program to solve Traveling Salesman Problem
// using Branch and Bound.
#include <bits/stdc++.h>
using namespace std;
const int N = 4;

// final_path[] stores the final solution ie, the
// path of the salesman.
int final_path[N+1];

// visited[] keeps track of the already visited nodes
// in a particular path
bool visited[N];

// Stores the final minimum weight of shortest tour.
int final_res = INT_MAX;

// Function to copy temporary solution to
// the final solution
void copyToFinal(int curr_path[])
{
    for (int i=0; i<N; i++)
        final_path[i] = curr_path[i];
    final_path[N] = curr_path[0];
}

// Function to find the minimum edge cost
// having an end at the vertex i
int firstMin(int adj[N][N], int i)
{
    int min = INT_MAX;
    for (int k=0; k<N; k++)
        if (adj[i][k]<min && i != k)
            min = adj[i][k];
    return min;
}

// function to find the second minimum edge cost
// having an end at the vertex i
int secondMin(int adj[N][N], int i)
{
    int first = INT_MAX, second = INT_MAX;
    for (int j=0; j<N; j++)
    {
        if (i == j)
            continue;

        if (adj[i][j] <= first)
        {
            second = first;
            first = adj[i][j];
        }
        else if (adj[i][j] <= second &&
                  adj[i][j] != first)
            second = adj[i][j];
    }
    return second;
}

// function that takes as arguments:
// curr_bound -> lower bound of the root node
// curr_weight-> stores the weight of the path so far
// level-> current level while moving in the search

```

```

// space tree
// curr_path[] -> where the solution is being stored which
// would later be copied to final_path[]
void TSPRec(int adj[N][N], int curr_bound, int curr_weight,
            int level, int curr_path[])
{
    // base case is when we have reached level N which
    // means we have covered all the nodes once
    if (level==N)
    {
        // check if there is an edge from last vertex in
        // path back to the first vertex
        if (adj[curr_path[level-1]][curr_path[0]] != 0)
        {
            // curr_res has the total weight of the
            // solution we got
            int curr_res = curr_weight +
                           adj[curr_path[level-1]][curr_path[0]];

            // Update final result and final path if
            // current result is better.
            if (curr_res < final_res)
            {
                copyToFinal(curr_path);
                final_res = curr_res;
            }
        }
        return;
    }

    // for any other level iterate for all vertices to
    // build the search space tree recursively
    for (int i=0; i<N; i++)
    {
        // Consider next vertex if it is not same (diagonal
        // entry in adjacency matrix and not visited
        // already)
        if (adj[curr_path[level-1]][i] != 0 &&
            visited[i] == false)
        {
            int temp = curr_bound;
            curr_weight += adj[curr_path[level-1]][i];

            // different computation of curr_bound for
            // level 2 from the other levels
            if (level==1)
                curr_bound -= ((firstMin(adj, curr_path[level-1]) +
                               firstMin(adj, i))/2);
            else
                curr_bound -= ((secondMin(adj, curr_path[level-1]) +
                               firstMin(adj, i))/2);

            // curr_bound + curr_weight is the actual lower bound
            // for the node that we have arrived on
            // If current lower bound < final_res, we need to explore
            // the node further
            if (curr_bound + curr_weight < final_res)
            {
                curr_path[level] = i;
                visited[i] = true;

                // call TSPRec for the next level
                TSPRec(adj, curr_bound, curr_weight, level+1,
                       curr_path);
            }

            // Else we have to prune the node by resetting
            // all changes to curr_weight and curr_bound
            curr_weight -= adj[curr_path[level-1]][i];
            curr_bound = temp;

            // Also reset the visited array
            memset(visited, false, sizeof(visited));
            for (int j=0; j<=level-1; j++)
        }
    }
}

```

```

        visited[curr_path[j]] = true;
    }
}

// This function sets up final_path[]
void TSP(int adj[N][N])
{
    int curr_path[N+1];

    // Calculate initial lower bound for the root node
    // using the formula 1/2 * (sum of first min +
    // second min) for all edges.
    // Also initialize the curr_path and visited array
    int curr_bound = 0;
    memset(curr_path, -1, sizeof(curr_path));
    memset(visited, 0, sizeof(curr_path));

    // Compute initial bound
    for (int i=0; i<N; i++)
        curr_bound += (firstMin(adj, i) +
                       secondMin(adj, i));

    // Rounding off the lower bound to an integer
    curr_bound = (curr_bound&1)? curr_bound/2 + 1 :
                           curr_bound/2;

    // We start at vertex 1 so the first vertex
    // in curr_path[] is 0
    visited[0] = true;
    curr_path[0] = 0;

    // Call to TSPRec for curr_weight equal to
    // 0 and level 1
    TSPRec(adj, curr_bound, 0, 1, curr_path);
}

// Driver code
int main()
{
    //Adjacency matrix for the given graph
    int adj[N][N] = { {0, 10, 15, 20},
                      {10, 0, 35, 25},
                      {15, 35, 0, 30},
                      {20, 25, 30, 0}
    };

    TSP(adj);

    printf("Minimum cost : %d\n", final_res);
    printf("Path Taken : ");
    for (int i=0; i<=N; i++)
        printf("%d ", final_path[i]);

    return 0;
}

```

[Run on IDE](#)

Output :

```

Minimum cost : 80
Path Taken : 0 1 3 2 0

```

Time Complexity: The worst case complexity of Branch and Bound remains same as that of the Brute Force clearly because in worst case, we may never get a chance to prune a node. Whereas, in practice it performs very well depending on the different instance of the TSP. The complexity also depends on the choice of the bounding function as they are the ones deciding how many nodes to be pruned.

References:

<http://lcm.csa.iisc.ernet.in/dsa/node187.html>

This article is contributed by **Anurag Rai**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Branch and Bound

Related Posts:

- [Branch And Bound | Set 4 \(Job Assignment Problem\)](#)
- [Branch and Bound | Set 5 \(N Queen Problem\)](#)
- [Branch and Bound | Set 3 \(8 puzzle Problem\)](#)
- [Branch and Bound | Set 2 \(Implementation of 0/1 Knapsack\)](#)
- [Branch and Bound | Set 1 \(Introduction with 0/1 Knapsack\)](#)

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 1 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

HANDBOOK OF ALGORITHMS

Section
Mathematics

Write an Efficient Method to Check if a Number is Multiple of 3 - GeeksforGeeks.pdf

Efficient way to multiply with 7 - GeeksforGeeks.pdf

Write a program to print all permutations of a given string - GeeksforGeeks.pdf

Lucky Numbers - GeeksforGeeks.pdf

Write a program to add two numbers in base 14 - GeeksforGeeks.pdf

Babylonian method for square root - GeeksforGeeks.pdf

Multiply two integers without using multiplication, division and bitwise operators, and no loops - GeeksforGeeks.pdf

Print all combinations of points that can compose a given number - GeeksforGeeks.pdf

Write your own Power without using multiplication(_) and division(_) operators - GeeksforGeeks.pdf

Program for Fibonacci numbers - GeeksforGeeks.pdf

Average of a stream of numbers - GeeksforGeeks.pdf

Count numbers that don't contain 3 - GeeksforGeeks.pdf

Magic Square - GeeksforGeeks.pdf

Sieve of Eratosthenes - GeeksforGeeks.pdf

Find day of the week for a given date - GeeksforGeeks.pdf

DFA based division - GeeksforGeeks.pdf

Generate integer from 1 to 7 with equal probability - GeeksforGeeks.pdf

Given a number, find the next smallest palindrome - GeeksforGeeks.pdf

Make a fair coin from a biased coin - GeeksforGeeks.pdf

Check divisibility by 7 - GeeksforGeeks.pdf

Find the largest multiple of 3 - GeeksforGeeks.pdf

Lexicographic rank of a string - GeeksforGeeks.pdf

Print all permutations in sorted (lexicographic) order - GeeksforGeeks.pdf

Shuffle a given array - GeeksforGeeks.pdf

Space and time efficient Binomial Coefficient - GeeksforGeeks.pdf

Reservoir Sampling - GeeksforGeeks.pdf

Pascal's Triangle - GeeksforGeeks.pdf

Select a random number from stream, with O(1) space - GeeksforGeeks.pdf

Find the largest multiple of 2, 3 and 5 - GeeksforGeeks.pdf

Efficient program to calculate e^x - GeeksforGeeks.pdf

Measure one litre using two vessels and infinite water supply.pdf

Efficient program to print all prime factors of a given number.pdf

Print all possible combinations of r elements in a given array of size n - GeeksforGeeks.pdf

Random number generator in arbitrary probability distribution fashion - GeeksforGeeks.pdf

How to check if a given number is Fibonacci number_ - GeeksforGeeks.pdf

Russian Peasant Multiplication - GeeksforGeeks.pdf

Count all possible groups of size 2 or 3 that have sum as multiple of 3 - GeeksforGeeks.pdf

C Program for Tower of Hanoi - GeeksQuiz.pdf

Horner's Method for Polynomial Evaluation - GeeksforGeeks.pdf

Count trailing zeroes in factorial of a number - GeeksforGeeks.pdf

Program for nth Catalan Number - GeeksforGeeks.pdf

Write a function that generates one of 3 numbers according to given probabilities - GeeksforGeeks.pdf

Find Excel column name from a given column number - GeeksforGeeks.pdf

Find next greater number with same set of digits - GeeksforGeeks.pdf

Count Possible Decodings of a given Digit Sequence - GeeksforGeeks.pdf

Calculate the angle between hour hand and minute hand - GeeksforGeeks.pdf

Count number of binary strings without consecutive 1's - GeeksforGeeks.pdf

Find the smallest number whose digits multiply to a given number n - GeeksforGeeks.pdf

Draw a circle without floating point arithmetic - GeeksQuiz.pdf

How to check if an instance of 8 puzzle is solvable_ - GeeksforGeeks.pdf

Birthday Paradox - GeeksforGeeks.pdf

Multiply two polynomials - GeeksforGeeks.pdf

Count Distinct Non-Negative Integer Pairs (x, y) that Satisfy the Inequality $x_x + y_y \leq n$ - GeeksforGeeks.pdf

Count ways to reach the n'th stair - GeeksforGeeks.pdf

Replace all '0' with '5' in an input Integer - GeeksQuiz.pdf

Program to add two polynomials - GeeksQuiz.pdf

Print first k digits of 1_n where n is a positive integer - GeeksQuiz.pdf

Given a number as a string, find the number of contiguous subsequences which recursively add up to 9 - GeeksQuiz.pdf

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Write an Efficient Method to Check if a Number is Multiple of 3

The very first solution that comes to our mind is the one that we learned in school. If sum of digits in a number is multiple of 3 then number is multiple of 3 e.g., for 612 sum of digits is 9 so it's a multiple of 3. But this solution is not efficient. You have to get all decimal digits one by one, add them and then check if sum is multiple of 3.

There is a pattern in binary representation of the number that can be used to find if number is a multiple of 3. If difference between count of odd set bits (Bits set at odd positions) and even set bits is multiple of 3 then is the number.

Example: 23 (00..10111)

- 1) Get count of all set bits at odd positions (For 23 it's 3).
- 2) Get count of all set bits at even positions (For 23 it's 1).
- 3) If difference of above two counts is a multiple of 3 then number is also a multiple of 3.

(For 23 it's 2 so 23 is not a multiple of 3)

Take some more examples like 21, 15, etc...

```
Algorithm: isMutlipleOf3(n)
1) Make n positive if n is negative.
2) If number is 0 then return 1
3) If number is 1 then return 0
4) Initialize: odd_count = 0, even_count = 0
5) Loop while n != 0
   a) If rightmost bit is set then increment odd count.
   b) Right-shift n by 1 bit
   c) If rightmost bit is set then increment even count.
   d) Right-shift n by 1 bit
6) return isMutlipleOf3(odd_count - even_count)
```

Proof:

Above can be proved by taking the example of 11 in decimal numbers. (In this context 11 in decimal numbers is same as 3 in binary numbers)

If difference between sum of odd digits and even digits is multiple of 11 then decimal number is multiple of 11. Let's see how.

Let's take the example of 2 digit numbers in decimal

$$AB = 11A - A + B = 11A + (B - A)$$

So if $(B - A)$ is a multiple of 11 then is AB.

Let us take 3 digit numbers.

$$ABC = 99A + A + 11B - B + C = (99A + 11B) + (A + C - B)$$

So if $(A + C - B)$ is a multiple of 11 then is $(A+C-B)$

Let us take 4 digit numbers now.

$$ABCD = 1001A + D + 11C - C + 999B + B - A$$

$$= (1001A - 999B + 11C) + (D + B - A - C)$$

So, if $(B + D - A - C)$ is a multiple of 11 then is $ABCD$.

This can be continued for all decimal numbers.

Above concept can be proved for 3 in binary numbers in the same way.

Time Complexity: $O(\log n)$

Program:

```
#include<stdio.h>

/* Function to check if n is a multiple of 3*/
int isMultipleOf3(int n)
{
    int odd_count = 0;
    int even_count = 0;

    /* Make no positive if +n is multiple of 3
       then is -n. We are doing this to avoid
       stack overflow in recursion*/
    if(n < 0)    n = -n;
    if(n == 0)   return 1;
    if(n == 1)   return 0;

    while(n)
    {
        /* If odd bit is set then
           increment odd counter */
        if(n & 1)
            odd_count++;
        n = n>>1;

        /* If even bit is set then
           increment even counter */
        if(n & 1)
            even_count++;
        n = n>>1;
    }

    return isMultipleOf3(abs(odd_count - even_count));
}

/* Program to test function isMultipleOf3 */
int main()
{
    int num = 23;
    if (isMultipleOf3(num))
        printf("num is multiple of 3");
    else
        printf("num is not a multiple of 3");
    getchar();
    return 0;
}
```

Run on IDE



GATE CS Corner Company Wise Coding Practice

[Bit Magic](#) [MathematicalAlgo](#)

Related Posts:

- XOR counts of 0s and 1s in binary representation
- Multiplying a variable with a constant without using multiplication operator
- Bitwise and (or &) of a range
- Calculate XOR from 1 to n.
- Multiples of 4 (An Interesting Method)
- Multiply a number with 10 without using multiplication operator
- Equal Sum and XOR
- Length of the Longest Consecutive 1s in Binary Representation

(Login to Rate and Mark)

3.7

Average Difficulty : 3.7/5.0
Based on 62 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Efficient way to multiply with 7

We can multiply a number by 7 using bitwise operator. First left shift the number by 3 bits (you will get $8n$) then subtract the original number from the shifted number and return the difference ($8n - n$).

Program:

```
# include<stdio.h>

int multiplyBySeven(unsigned int n)
{
    /* Note the inner bracket here. This is needed
       because precedence of '-' operator is higher
       than '<<' */
    return ((n<<3) - n);
}

/* Driver program to test above function */
int main()
{
    unsigned int n = 4;
    printf("%u", multiplyBySeven(n));

    getchar();
    return 0;
}
```

[Run on IDE](#)

Time Complexity: O(1)

Space Complexity: O(1)

Note: Works only for positive integers.

Same concept can be used for fast multiplication by 9 or other numbers.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

Bit Magic Bit Magic MathematicalAlgo

Related Posts:

- XOR counts of 0s and 1s in binary representation
- Multiplying a variable with a constant without using multiplication operator
- Bitwise and (or &) of a range
- Calculate XOR from 1 to n.
- Multiples of 4 (An Interesting Method)
- Multiply a number with 10 without using multiplication operator
- Equal Sum and XOR
- Length of the Longest Consecutive 1s in Binary Representation

(Login to Rate and Mark)

1.7

Average Difficulty : **1.7/5.0**
Based on **38** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Write a program to print all permutations of a given string

A permutation, also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has $n!$ permutation.

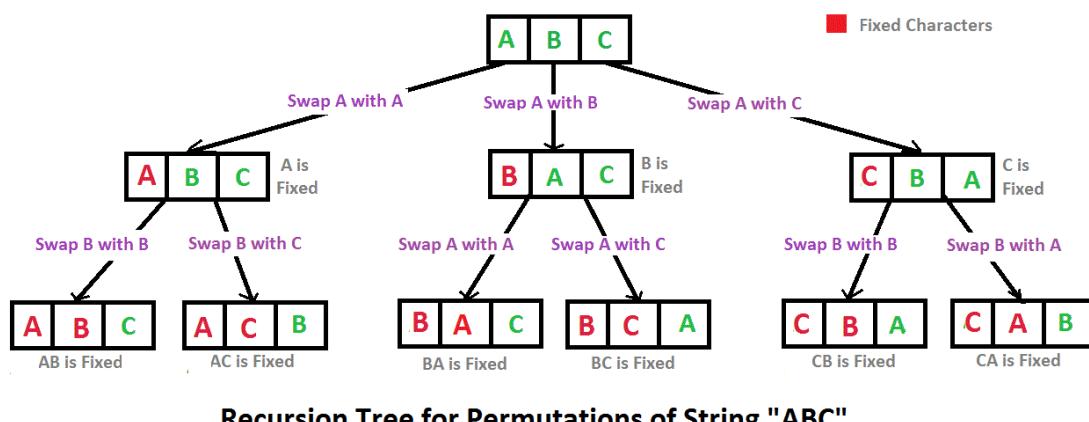
Source: Mathword(<http://mathworld.wolfram.com/Permutation.html>)

Below are the permutations of string ABC.

ABC ACB BAC BCA CBA CAB

We strongly recommend that you click here and practice it, before moving on to the solution.

Here is a solution that is used as a basis in backtracking.



```
// C program to print all permutations with duplicates allowed
#include <stdio.h>
#include <string.h>

/* Function to swap values at two pointers */
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

```

*x = *y;
*y = temp;
}

/* Function to print permutations of string
This function takes three parameters:
1. String
2. Starting index of the string
3. Ending index of the string. */
void permute(char *a, int l, int r)
{
    int i;
    if (l == r)
        printf("%s\n", a);
    else
    {
        for (i = l; i <= r; i++)
        {
            swap((a+l), (a+i));
            permute(a, l+1, r);
            swap((a+l), (a+i)); //backtrack
        }
    }
}

/* Driver program to test above functions */
int main()
{
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to print all permutations with
# duplicates allowed

def toString(List):
    return ''.join(List)

# Function to print permutations of string
# This function takes three parameters:
# 1. String
# 2. Starting index of the string
# 3. Ending index of the string.
def permute(a, l, r):
    if l==r:
        print toString(a)
    else:
        for i in xrange(l,r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l] # backtrack

# Driver program to test the above function
string = "ABC"
n = len(string)
a = list(string)
permute(a, 0, n-1)

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

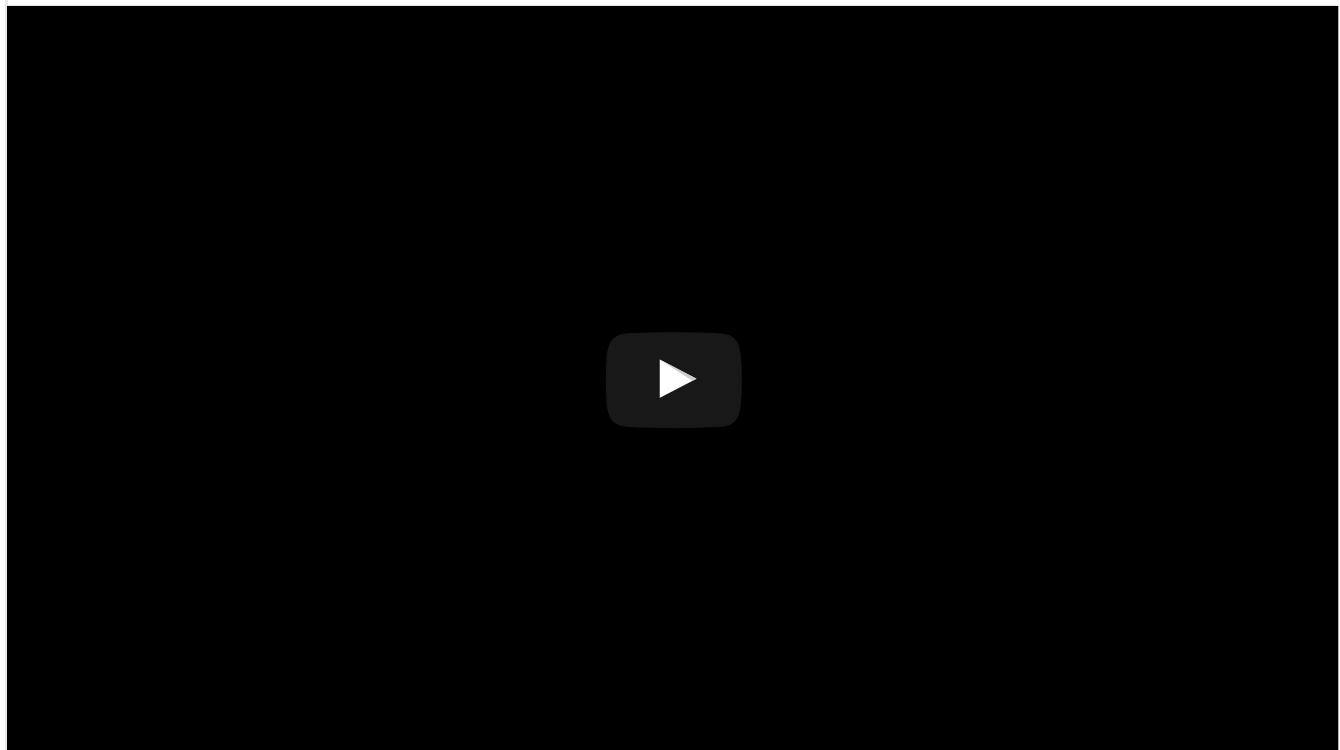
```
ABC  
ACB  
BAC  
BCA  
CBA  
CAB
```

Algorithm Paradigm: Backtracking

Time Complexity: $O(n*n!)$ Note that there are $n!$ permutations and it requires $O(n)$ time to print a permutation.

Note : The above solution prints duplicate permutations if there are repeating characters in input string. Please see below link for a solution that prints only distinct permutations even if there are duplicates in input.

[Print all distinct permutations of a given string with duplicates.](#)



Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Combinatorial Strings Backtracking MathematicalAlgo permutation

Related Posts:

- Sum of all numbers that can be formed with permutations of n digits
- Longest common subsequence with permutations allowed
- Number of ways to make mobile lock pattern
- Count permutations that produce positive result
- Find all distinct subsets of a given set
- Heap's Algorithm for generating permutations
- All permutations of a string using iteration
- Permutations of a given string using STL

(Login to Rate and Mark)

3.4

Average Difficulty : 3.4/5.0
Based on 180 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Lucky Numbers

Lucky numbers are subset of integers. Rather than going into much theory, let us see the process of arriving at lucky numbers,

Take the set of integers

1,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,.....

First, delete every second number, we get following reduced set.

1,3,5,7,9,11,13,15,17,19,.....

Now, delete every third number, we get

1, 3, 7, 9, 13, 15, 19,.....

Continue this process indefinitely.....

Any number that does NOT get deleted due to above process is called “lucky”.

Therefore, set of lucky numbers is 1, 3, 7, 13,.....

Now, given an integer ‘n’, write a function to say whether this number is lucky or not.

```
bool isLucky(int n)
```

Algorithm:

Before every iteration, if we calculate position of the given no, then in a given iteration, we can determine if the no will be deleted. Suppose calculated position for the given no. is P before some iteration, and each Ith no. is going to be removed in this iteration, if $P < I$ then input no is lucky, if P is such that $P \% I == 0$ (I is a divisor of P), then input no is not lucky. **Recursive Way:**

```
#include <stdio.h>
#define bool int

/* Returns 1 if n is a lucky no. otherwise returns 0*/
bool isLucky(int n)
{
    static int counter = 2;

    /*variable next_position is just for readability of
       the program we can remove it and use n only */
    int next_position = n;
    if(counter > n)
        return 1;
    if(n%counter == 0)
        return 0;

    /*calculate next position of input no*/
    next_position -= next_position/counter;
```

```

    counter++;
    return isLucky(next_position);
}

/*Driver function to test above function*/
int main()
{
    int x = 5;
    if( isLucky(x) )
        printf("%d is a lucky no.", x);
    else
        printf("%d is not a lucky no.", x);
    getchar();
}

```

[Run on IDE](#)**Example:**

Let's us take an example of 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,15,17,18,19,20,21,.....

1,3,5,7,9,11,13,15,17,19,.....

1,3,7,9,13,15,19,.....

1,3,7,13,15,19,.....

1,3,7,13,19,.....

In next step every 6th no .in sequence will be deleted. 19 will not be deleted after this step because position of 19 is 5th after this step. Therefore, 19 is lucky. Let's see how above C code finds out:

Current function call	Position after this call	Counter for next call	Next Call
isLucky(19)	10	3	isLucky(10)
isLucky(10)	7	4	isLucky(7)
isLucky(7)	6	5	isLucky(6)
isLucky(6)	5	6	isLucky(5)

When isLucky(6) is called, it returns 1 (because counter > n).

Iterative Way:

Please see [this](#) comment for another simple and elegant implementation of the above algorithm.

Please write comments if you find any bug in the given programs or other ways to solve the same problem.



GATE CS Corner Company Wise Coding Practice

[Mathematical](#)
[C Program for Lucky Numbers](#)
[isLucky](#)
[Lucky Number](#)
[Lucky Numbers](#)
[MathematicalAlgo](#)
[Print Lucky numbers](#)
[Program for lucky numbers](#)
[Programs for numbers](#)

Related Posts:

- [Nth character in Concatenated Decimal String](#)
- [Find the highest occurring digit in prime numbers in a range](#)
- [Smallest number to multiply to convert floating point to natural](#)
- [Generate all palindromic numbers less than n](#)
- [Number of sextuplets \(or six values\) that satisfy an equation](#)
- [Circular primes less than n](#)
- [Sphenic Number](#)
- [Minimum sum of two numbers formed from digits of an array](#)

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 33 vote(s)


[Add to TODO List](#)
[Mark as DONE](#)

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Write a program to add two numbers in base 14

Asked by Anshya.

Below are the different ways to add base 14 numbers.

Method 1

Thanks to Raj for suggesting this method.

1. Convert both i/p base 14 numbers to base 10.
2. Add numbers.
3. Convert the result back to base 14.

Method 2

Just add the numbers in base 14 in same way we add in base 10. Add numerals of both numbers one by one from right to left. If there is a carry while adding two numerals, consider the carry for adding next numerals.

Let us consider the presentation of base 14 numbers same as hexadecimal numbers

```

A --> 10
B --> 11
C --> 12
D --> 13

```

Example:

```

num1 =      1  2  A
num2 =      C  D  3

```

1. Add A and 3, we get 13(D). Since 13 is smaller than 14, carry becomes 0 and resultant numeral becomes D
2. Add 2, D and carry(0). we get 15. Since 15 is greater than 14, carry becomes 1 and resultant numeral is $15 - 14 = 1$
3. Add 1, C and carry(1). we get 14. Since 14 is greater than 13, carry becomes 1 and resultant numeral is $14 - 14 = 0$

Finally, there is a carry, so 1 is added as leftmost numeral and the result becomes 101D

Implementation of Method 2

```

#include <stdio.h>
#include <stdlib.h>
#define bool int

int getNumeralValue(char );
char getNumeral(int );

/* Function to add two numbers in base 14 */
char *sumBase14(char *num1, char *num2)
{
    int l1 = strlen(num1);
    int l2 = strlen(num2);
    char *res;
    int i;
    int nml1, nml2, res_nml;
    bool carry = 0;

    if(l1 != l2)
    {
        printf("Function doesn't support numbers of different"
               " lengths. If you want to add such numbers then"
               " prefix smaller number with required no. of zeroes");
        getchar();
        assert(0);
    }

    /* Note the size of the allocated memory is one
       more than i/p lengths for the cases where we
       have carry at the last like adding D1 and A1 */
    res = (char *)malloc(sizeof(char)*(l1 + 1));

    /* Add all numerals from right to left */
    for(i = l1-1; i >= 0; i--)
    {
        /* Get decimal values of the numerals of
           i/p numbers*/
        nml1 = getNumeralValue(num1[i]);
        nml2 = getNumeralValue(num2[i]);

        /* Add decimal values of numerals and carry */
        res_nml = carry + nml1 + nml2;

        /* Check if we have carry for next addition
           of numerals */
        if(res_nml >= 14)
        {
            carry = 1;
            res_nml -= 14;
        }
        else
        {
            carry = 0;
        }
        res[i+1] = getNumeral(res_nml);
    }

    /* if there is no carry after last iteration
       then result should not include 0th character
       of the resultant string */
    if(carry == 0)
        return (res + 1);

    /* if we have carry after last iteration then
       result should include 0th character */
    res[0] = '1';
    return res;
}

/* Function to get value of a numeral
   For example it returns 10 for input 'A'
   1 for '1', etc */
int getNumeralValue(char num)
{
    if( num >= '0' && num <= '9')

```

```

        return (num - '0');
if( num >= 'A' && num <= 'D')
    return (num - 'A' + 10);

/* If we reach this line caller is giving
   invalid character so we assert and fail*/
assert(0);
}

/* Function to get numeral for a value.
   For example it returns 'A' for input 10
   '1' for 1, etc */
char getNumeral(int val)
{
    if( val >= 0 && val <= 9)
        return (val + '0');
    if( val >= 10 && val <= 14)
        return (val + 'A' - 10);

    /* If we reach this line caller is giving
       invalid no. so we assert and fail*/
    assert(0);
}

/*Driver program to test above functions*/
int main()
{
    char *num1 = "DC2";
    char *num2 = "0A3";

    printf("Result is %s", sumBase14(num1, num2));
    getchar();
    return 0;
}

```

[Run on IDE](#)**Notes:**

Above approach can be used to add numbers in any base. We don't have to do string operations if base is smaller than 10.

You can try extending the above program for numbers of different lengths.

Please comment if you find any bug in the program or a better approach to do the same.

**China huge tile factory**

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

**GATE CS Corner Company Wise Coding Practice**

[Mathematical](#)[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.5

Average Difficulty : **2.5/5.0**
Based on **12** vote(s)

[Add to TODO List](#)[Mark as DONE](#)

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Babylonian method for square root

Algorithm:

This method can be derived from (but predates) Newton–Raphson method.

- 1 Start with an arbitrary positive start value x (the closer to the root, the better).
- 2 Initialize $y = 1$.
3. Do following until desired approximation is achieved.
 - a) Get the next approximation for root using average of x and y
 - b) Set $y = n/x$

Implementation:

```
/*Returns the square root of n. Note that the function */
float squareRoot(float n)
{
    /*We are using n itself as initial approximation
     This can definitely be improved */
    float x = n;
    float y = 1;
    float e = 0.000001; /* e decides the accuracy level*/
    while(x - y > e)
    {
        x = (x + y)/2;
        y = n/x;
    }
    return x;
}

/* Driver program to test above function*/
int main()
{
    int n = 50;
    printf ("Square root of %d is %f", n, squareRoot(n));
    getchar();
}
```

[Run on IDE](#)

Example:

```
n = 4 /*n itself is used for initial approximation*/
Initialize x = 4, y = 1
Next Approximation x = (x + y)/2 (= 2.500000),
y = n/x  (=1.600000)
Next Approximation x = 2.050000,
y = 1.951220
```

```
Next Approximation x = 2.000610,
y = 1.999390
Next Approximation x = 2.000000,
y = 2.000000
Terminate as (x - y) > e now.
```

If we are sure that n is a perfect square, then we can use following method. The method can go in infinite loop for non-perfect-square numbers. For example, for 3 the below while loop will never terminate.

```
/*Returns the square root of n. Note that the function
   will not work for numbers which are not perfect squares*/
unsigned int squareRoot(int n)
{
    int x = n;
    int y = 1;
    while(x > y)
    {
        x = (x + y)/2;
        y = n/x;
    }
    return x;
}

/* Driver program to test above function*/
int main()
{
    int n = 49;
    printf (" root of %d is %d", n, squareRoot(n));
    getchar();
}
```

[Run on IDE](#)

References;

http://en.wikipedia.org/wiki/Square_root

http://en.wikipedia.org/wiki/Babylonian_method#Babylonian_method

Asked by Snehah

Please write comments if you find any bug in the above program/algorithm, or if you want to share more information about Babylonian method.



GATE CS Corner Company Wise Coding Practice

[Mathematical](#)[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

3.1

Average Difficulty : 3.1/5.0
Based on 21 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Multiply two integers without using multiplication, division and bitwise operators, and no loops

Asked by [Kapil](#)

By making use of recursion, we can multiply two integers with the given constraints.

To multiply x and y, recursively add x y times.

Thanks to [geek4u](#) for suggesting this method.

```
#include<stdio.h>
/* function to multiply two numbers x and y*/
int multiply(int x, int y)
{
    /* 0 multiplied with anything gives 0 */
    if(y == 0)
        return 0;

    /* Add x one by one */
    if(y > 0 )
        return (x + multiply(x, y-1));

    /* the case where y is negative */
    if(y < 0 )
        return -multiply(x, -y);
}

int main()
{
    printf("\n %d", multiply(5, -11));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Time Complexity: O(y) where y is the second argument to function multiply().

Please write comments if you find any of the above code/algorithm incorrect, or find better ways to solve the same problem.



Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now



HARVARD
UNIVERSITY

GATE CS Corner Company Wise Coding Practice

Mathematical

MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

1.8

Average Difficulty : 1.8/5.0
Based on 25 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)[GATE CS](#)[Placements](#)[GeeksQuiz](#)[Login/Register](#)

Print all combinations of points that can compose a given number

You can win three kinds of basketball points, 1 point, 2 points, and 3 points. Given a total score n, print out all the combination to compose n.

Examples:

For n = 1, the program should print following:

1

For n = 2, the program should print following:

1 1

2

For n = 3, the program should print following:

1 1 1

1 2

2 1

3

For n = 4, the program should print following:

1 1 1 1

1 1 2

1 2 1

1 3

2 1 1

2 2

3 1

and so on ...

Algorithm:

At first position we can have three numbers 1 or 2 or 3.

First put 1 at first position and recursively call for n-1.

Then put 2 at first position and recursively call for n-2.

Then put 3 at first position and recursively call for n-3.

If n becomes 0 then we have formed a combination that compose n, so print the current combination.

Below is a generalized implementation. In the below implementation, we can change MAX_POINT if there are higher points (more than 3) in the basketball game.

```

#define MAX_POINT 3
#define ARR_SIZE 100
#include<stdio.h>

/* Utility function to print array arr[] */
void printArray(int arr[], int arr_size);

/* The function prints all combinations of numbers 1, 2, ...MAX_POINT
   that sum up to n.
   i is used in recursion keep track of index in arr[] where next
   element is to be added. Initial value of i must be passed as 0 */
void printCompositions(int n, int i)
{
    /* array must be static as we want to keep track
       of values stored in arr[] using current calls of
       printCompositions() in function call stack*/
    static int arr[ARR_SIZE];

    if (n == 0)
    {
        printArray(arr, i);
    }
    else if(n > 0)
    {
        int k;
        for (k = 1; k <= MAX_POINT; k++)
        {
            arr[i]= k;
            printCompositions(n-k, i+1);
        }
    }
}

/* UTILITY FUNCTIONS */
/* Utility function to print array arr[] */
void printArray(int arr[], int arr_size)
{
    int i;
    for (i = 0; i < arr_size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* Driver function to test above functions */
int main()
{
    int n = 5;
    printf("Differnt compositions formed by 1, 2 and 3 of %d are\n", n);
    printCompositions(n, 0);
    getchar();
    return 0;
}

```

[Run on IDE](#)

Asked by Aloe

Please write comments if you find any bug in above code/algorithm, or find other ways to solve the same problem.

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 11 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Write you own Power without using multiplication(*) and division(/) operators

Method 1 (Using Nested Loops)

We can calculate power by using repeated addition.

For example to calculate 5^6 .

- 1) First 5 times add 5, we get 25. (5^2)
- 2) Then 5 times add 25, we get 125. (5^3)
- 3) Then 5 time add 125, we get 625 (5^4)
- 4) Then 5 times add 625, we get 3125 (5^5)
- 5) Then 5 times add 3125, we get 15625 (5^6)

```
/* Works only if a >= 0 and b >= 0 */
int pow(int a, int b)
{
    if (b == 0)
        return 1;
    int answer = a;
    int increment = a;
    int i, j;
    for(i = 1; i < b; i++)
    {
        for(j = 1; j < a; j++)
        {
            answer += increment;
        }
        increment = answer;
    }
    return answer;
}

/* driver program to test above function */
int main()
{
    printf("\n %d", pow(5, 3));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Method 2 (Using Recursion)

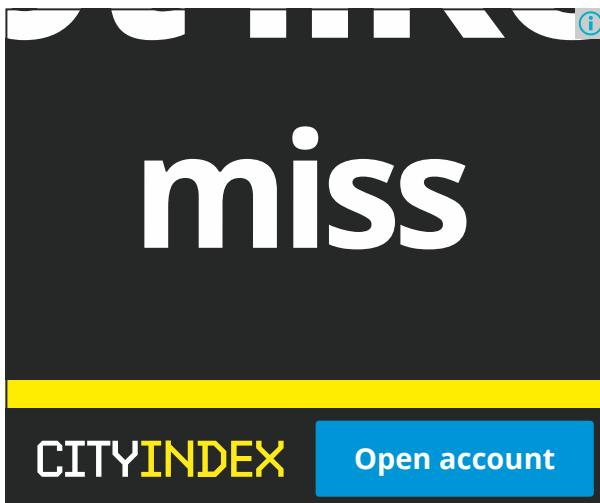
Recursively add a to get the multiplication of two numbers. And recursively multiply to get a raise to the power b.

```
#include<stdio.h>
/* A recursive function to get a^b
   Works only if a >= 0 and b >= 0 */
int pow(int a, int b)
```

```
{  
    if(b)  
        return multiply(a, pow(a, b-1));  
    else  
        return 1;  
}  
  
/* A recursive function to get x*y */  
int multiply(int x, int y)  
{  
    if(y)  
        return (x + multiply(x, y-1));  
    else  
        return 0;  
}  
  
/* driver program to test above functions */  
int main()  
{  
    printf("\n %d", pow(5, 3));  
    getchar();  
    return 0;  
}
```

[Run on IDE](#)

Please write comments if you find any bug in above code/algorith, or find other ways to solve the same problem.



GATE CS Corner Company Wise Coding Practice

[Divide and Conquer](#) [Mathematical](#) [MathematicalAlgo](#)

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 19 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Program for Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values

$$F_0 = 0 \text{ and } F_1 = 1.$$

We strongly recommend that you click here and practice it, before moving on to the solution.

Write a function `int fib(int n)` that returns F_n . For example, if $n = 0$, then `fib()` should return 0. If $n = 1$, then it should return 1. For $n > 1$, it should return $F_{n-1} + F_{n-2}$

Output:34

Following are different methods to get the nth Fibonacci number.

Method 1 (Use recursion)

A simple method that is a direct recursive implementation mathematical recurrence relation given above.

```
//Fibonacci Series using Recursion
#include<stdio.h>
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}

int main ()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
}
```

```

    return 0;
}

```

[Run on IDE](#)

Java

```

//Fibonacci Series using Recursion
class fibonacci
{
    static int fib(int n)
    {
        if (n <= 1)
            return n;
        return fib(n-1) + fib(n-2);
    }

    public static void main (String args[])
    {
        int n = 9;
        System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# Function for nth Fibonacci number

def Fibonacci(n):
    if n<0:
        print("Incorrect input")
    # First Fibonacci number is 0
    elif n==1:
        return 0
    # Second Fibonacci number is 1
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

# Driver Program

print(Fibonacci(9))

#This code is contributed by Saket Modi

```

[Run on IDE](#)

Output

21

Time Complexity: $T(n) = T(n-1) + T(n-2)$ which is exponential.

We can observe that this implementation does a lot of repeated work (see the following recursion tree). So this is a bad implementation for nth Fibonacci number.

```

        fib(5)
        /           \
    fib(4)           fib(3)
    /   \           /   \
fib(3)   fib(2)   fib(2)   fib(1)
/   \   /   \   /   \
fib(2)   fib(1)   fib(1)   fib(0)   fib(1)   fib(0)
/   \
fib(1)   fib(0)

```

Extra Space: O(n) if we consider the function call stack size, otherwise O(1).

Method 2 (Use Dynamic Programming)

We can avoid the repeated work done in the method 1 by storing the Fibonacci numbers calculated so far.

```

//Fibonacci Series using Dynamic Programming
#include<stdio.h>

int fib(int n)
{
    /* Declare an array to store Fibonacci numbers. */
    int f[n+1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for (i = 2; i <= n; i++)
    {
        /* Add the previous 2 numbers in the series
         * and store it */
        f[i] = f[i-1] + f[i-2];
    }

    return f[n];
}

int main ()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
    return 0;
}

```

Run on IDE

Java

```

// Fibonacci Series using Dynamic Programming
class fibonacci
{
    static int fib(int n)
    {
        /* Declare an array to store Fibonacci numbers. */
        int f[] = new int[n+1];
        int i;

        /* 0th and 1st number of the series are 0 and 1*/
        f[0] = 0;
        f[1] = 1;
    }
}

```

```

for (i = 2; i <= n; i++)
{
    /* Add the previous 2 numbers in the series
       and store it */
    f[i] = f[i-1] + f[i-2];
}

return f[n];
}

public static void main (String args[])
{
    int n = 9;
    System.out.println(fib(n));
}
*/
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Python

```

# Function for nth fibonacci number - Dynamic Programming
# Taking 1st two fibonacci numbers as 0 and 1

FibArray = [0,1]

def fibonacci(n):
    if n<0:
        print("Incorrect input")
    elif n<=len(FibArray):
        return FibArray[n-1]
    else:
        temp_fib = fibonacci(n-1)+fibonacci(n-2)
        FibArray.append(temp_fib)
        return temp_fib

# Driver Program

print(fibonacci(9))

#This code is contributed by Saket Modi

```

Run on IDE

Output:

21

Time Complexity: O(n)

Extra Space: O(n)

Method 3 (Space Optimized Method 2)

We can optimize the space used in method 2 by storing the previous two numbers only because that is all we need to get the next Fibonacci number in series.

```

//Fibonacci Series using Space Optimized Method
#include<stdio.h>
int fib(int n)

```

```

{
    int a = 0, b = 1, c, i;
    if( n == 0)
        return a;
    for (i = 2; i <= n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

int main ()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
    return 0;
}

```

[Run on IDE](#)

Python

```

# Function for nth fibonacci number - Space Optimisataion
# Taking 1st two fibonacci numbers as 0 and 1

def fibonacci(n):
    a = 0
    b = 1
    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return a
    elif n == 1:
        return b
    else:
        for i in range(2,n):
            c = a + b
            a = b
            b = c
        return b

# Driver Program

print(fibonacci(9))

#This code is contributed by Saket Modi

```

[Run on IDE](#)

Time Complexity: O(n)

Extra Space: O(1)

Method 4 (Using power of the matrix {{1,1},{1,0}})

This another O(n) which relies on the fact that if we n times multiply the matrix $M = \begin{Bmatrix} 1 & 1 \\ 1 & 0 \end{Bmatrix}$ to itself (in other words calculate $\text{power}(M, n)$), then we get the $(n+1)$ th Fibonacci number as the element at row and column (0, 0) in the resultant matrix.

The matrix representation gives the following closed expression for the Fibonacci numbers:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

```
#include <stdio.h>

/* Helper function that multiplies 2 matrices F and M of size 2*2, and
   puts the multiplication result back to F[][] */
void multiply(int F[2][2], int M[2][2]);

/* Helper function that calculates F[][] raise to the power n and puts the
   result in F[][]
   Note that this function is designed only for fib() and won't work as general
   power function */
void power(int F[2][2], int n);

int fib(int n)
{
    int F[2][2] = {{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);

    return F[0][0];
}

void multiply(int F[2][2], int M[2][2])
{
    int x = F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y = F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z = F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w = F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

void power(int F[2][2], int n)
{
    int i;
    int M[2][2] = {{1,1},{1,0}};

    // n - 1 times multiply the matrix to {{1,0},{0,1}}
    for (i = 2; i <= n; i++)
        multiply(F, M);
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Java

```
class fibonacci
{
```

```

static int fib(int n)
{
    int F[][] = new int[][]{{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);

    return F[0][0];
}

/* Helper function that multiplies 2 matrices F and M of size 2*2, and
   puts the multiplication result back to F[][] */
static void multiply(int F[][], int M[][])
{
    int x = F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y = F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z = F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w = F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

/* Helper function that calculates F[][] raise to the power n and puts the
   result in F[][].
   Note that this function is designed only for fib() and won't work as general
   power function */
static void power(int F[][], int n)
{
    int i;
    int M[][] = new int[][]{{1,1},{1,0}};

    // n - 1 times multiply the matrix to {{1,0},{0,1}}
    for (i = 2; i <= n; i++)
        multiply(F, M);
}

/* Driver program to test above function */
public static void main (String args[])
{
    int n = 9;
    System.out.println(fib(n));
}
*/
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Time Complexity: O(n)

Extra Space: O(1)

Method 5 (Optimized Method 4)

The method 4 can be optimized to work in O(Logn) time complexity. We can do recursive multiplication to get power(M, n) in the previous method (Similar to the optimization done in [this](#) post)

```
#include <stdio.h>

void multiply(int F[2][2], int M[2][2]);

void power(int F[2][2], int n);
```

```

/* function that returns nth Fibonacci number */
int fib(int n)
{
    int F[2][2] = {{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);
    return F[0][0];
}

/* Optimized version of power() in method 4 */
void power(int F[2][2], int n)
{
    if( n == 0 || n == 1)
        return;
    int M[2][2] = {{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
}

void multiply(int F[2][2], int M[2][2])
{
    int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d", fib(9));
    getchar();
    return 0;
}

```

[Run on IDE](#)

Java

```

//Fibonacci Series using Optimized Method
class fibonacci
{
    /* function that returns nth Fibonacci number */
    static int fib(int n)
    {
        int F[][] = new int[][]{{1,1},{1,0}};
        if (n == 0)
            return 0;
        power(F, n-1);

        return F[0][0];
    }

    static void multiply(int F[][], int M[][])
    {
        int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
        int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
        int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
        int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];
    }
}

```

```

F[0][0] = x;
F[0][1] = y;
F[1][0] = z;
F[1][1] = w;
}

/* Optimized version of power() in method 4 */
static void power(int F[][], int n)
{
    if( n == 0 || n == 1)
        return;
    int M[][] = new int[][]{{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
}

/* Driver program to test above function */
public static void main (String args[])
{
    int n = 9;
    System.out.println(fib(n));
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Time Complexity: O(Logn)

Extra Space: O(Logn) if we consider the function call stack size, otherwise O(1).

Method 6 (O(Log n) Time)

Below is one more interesting recurrence formula that can be used to find n'th Fibonacci Number in O(Log n) time.

If n is even then k = n/2:
 $F(n) = [2*F(k-1) + F(k)]*F(k)$

If n is odd then k = (n + 1)/2
 $F(n) = F(k)*F(k) + F(k-1)*F(k-1)$

How does this formula work?

The formula can be derived from above matrix equation.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Taking determinant on both sides, we get

$$(-1)^n = F_{n+1}F_{n-1} - F_n^2$$

Moreover, since $A^n A^m = A^{n+m}$ for any square matrix A, the following identities can be derived (they are obtained from two different coefficients of the matrix product)

$$F_m F_n + F_{m-1} F_{n-1} = F_{m+n-1}$$

By putting n = n+1,

$$F_m F_{n+1} + F_{m-1} F_n = F_{m+n}$$

Putting $m = n$

$$F_{2n-1} = F_n^2 + F_{n-1}^2$$

$$F_{2n} = (F_{n-1} + F_{n+1})F_n = (2F_{n-1} + F_n)F_n \text{ (Source: } \text{Wiki})$$

To get the formula to be proved, we simply need to do following

If n is even, we can put $k = n/2$

If n is odd, we can put $k = (n+1)/2$

Below is C++ implementation of above idea.

```
// C++ Program to find n'th fibonacci Number in
// with O(Log n) arithmetic operations
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

// Create an array for memoization
int f[MAX] = {0};

// Returns n'th fuibonacci number using table f[]
int fib(int n)
{
    // Base cases
    if (n == 0)
        return 0;
    if (n == 1 || n == 2)
        return (f[n] = 1);

    // If fib(n) is already computed
    if (f[n])
        return f[n];

    int k = (n & 1)? (n+1)/2 : n/2;

    // Applying above formula [Note value n&1 is 1
    // if n is odd, else 0.
    f[n] = (n & 1)? (fib(k)*fib(k) + fib(k-1)*fib(k-1))
                  : (2*fib(k-1) + fib(k))*fib(k);

    return f[n];
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d ", fib(n));
    return 0;
}
```

Run on IDE

Output :

34

Time complexity of this solution is $O(\log n)$ as we divide the problem to half in every recursive call.

This method is contributed by Chirag Agarwal.

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

References:

http://en.wikipedia.org/wiki/Fibonacci_number
<http://www.ics.uci.edu/~eppstein/161/960109.html>

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Mathematical](#) [Dynamic Programming](#) [Fibonacci](#) [MathematicalAlgo](#) [series](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.1 Average Difficulty : 3.1/5.0
Based on 83 vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Average of a stream of numbers

Difficulty Level: Rookie

Given a stream of numbers, print average (or mean) of the stream at every point. For example, let us consider the stream as 10, 20, 30, 40, 50, 60, ...

```
Average of 1 numbers is 10.00
Average of 2 numbers is 15.00
Average of 3 numbers is 20.00
Average of 4 numbers is 25.00
Average of 5 numbers is 30.00
Average of 6 numbers is 35.00
.....
```

To print mean of a stream, we need to find out how to find average when a new number is being added to the stream. To do this, all we need is count of numbers seen so far in the stream, previous average and new number. Let n be the count, $prev_avg$ be the previous average and x be the new number being added. The average after including x number can be written as $(prev_avg * n + x) / (n+1)$.

```
#include <stdio.h>

// Returns the new average after including x
float getAvg(float prev_avg, int x, int n)
{
    return (prev_avg*n + x)/(n+1);
}

// Prints average of a stream of numbers
void streamAvg(float arr[], int n)
{
    float avg = 0;
    for(int i = 0; i < n; i++)
    {
        avg = getAvg(avg, arr[i], i);
        printf("Average of %d numbers is %f \n", i+1, avg);
    }
    return;
}

// Driver program to test above functions
int main()
{
    float arr[] = {10, 20, 30, 40, 50, 60};
    int n = sizeof(arr)/sizeof(arr[0]);
    streamAvg(arr, n);

    return 0;
}
```

[Run on IDE](#)

The above function getAvg() can be optimized using following changes. We can avoid the use of prev_avg and number of elements by using static variables (Assuming that only this function is called for average of stream). Following is the optimized version.

```
#include <stdio.h>

// Returns the new average after including x
float getAvg (int x)
{
    static int sum, n;

    sum += x;
    return (((float)sum)/++n);
}

// Prints average of a stream of numbers
void streamAvg(float arr[], int n)
{
    float avg = 0;
    for(int i = 0; i < n; i++)
    {
        avg = getAvg(arr[i]);
        printf("Average of %d numbers is %f \n", i+1, avg);
    }
    return;
}

// Driver program to test above functions
int main()
{
    float arr[] = {10, 20, 30, 40, 50, 60};
    int n = sizeof(arr)/sizeof(arr[0]);
    streamAvg(arr, n);

    return 0;
}
```

[Run on IDE](#)

Thanks to [Abhijeet Deshpande](#) for suggesting this optimized version.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



[Mathematical](#)[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

1.5

Average Difficulty : 1.5/5.0
Based on 18 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count numbers that don't contain 3

Given a number n, write a function that returns count of numbers from 1 to n that don't contain digit 3 in their decimal representation.

Examples:

Input: n = 10

Output: 9

Input: n = 45

Output: 31

// Numbers 3, 13, 23, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 43 contain digit 3.

Input: n = 578

Output: 385

We strongly recommend that you click here and practice it, before moving on to the solution.

Solution:

We can solve it recursively. Let count(n) be the function that counts such numbers.

```
'msd' --> the most significant digit in n
'd'   --> number of digits in n.

count(n) = n if n < 3

count(n) = n - 1 if 3 <= n < 10

count(n) = count(msd) * count(10^(d-1) - 1) +
           count(msd) +
           count(n % (10^(d-1)))
           if n > 10 and msd is not 3

count(n) = count(msd * (10^(d-1)) - 1)
           if n > 10 and msd is 3
```

Let us understand the solution with $n = 578$.

$\text{count}(578) = 4 * \text{count}(99) + 4 + \text{count}(78)$

The middle term 4 is added to include numbers 100, 200, 400 and 500.

Let us take $n = 35$ as another example.

$\text{count}(35) = \text{count}(3 * 10 - 1) = \text{count}(29)$

```
#include <stdio.h>

/* returns count of numbers which are in range from 1 to n and don't contain 3
   as a digit */
int count(int n)
{
    // Base cases (Assuming n is not negative)
    if (n < 3)
        return n;
    if (n >= 3 && n < 10)
        return n-1;

    // Calculate  $10^{(d-1)}$  (10 raise to the power d-1) where d is
    // number of digits in n. po will be 100 for n = 578
    int po = 1;
    while (n/po > 9)
        po = po*10;

    // find the most significant digit (msd is 5 for 578)
    int msd = n/po;

    if (msd != 3)
        // For 578, total will be  $4 * \text{count}(10^2 - 1) + 4 + \text{count}(78)$ 
        return count(msd)*count(po - 1) + count(msd) + count(n%po);
    else
        // For 35, total will be equal to count(29)
        return count(msd*po - 1);
}

// Driver program to test above function
int main()
{
    printf ("%d ", count(578));
    return 0;
}
```

Run on IDE

Output:

385

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Mathematical

MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4.1

Average Difficulty : 4.1/5.0
Based on 22 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Magic Square

A **magic square** of order n is an arrangement of n^2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A magic square contains the integers from 1 to n^2 .

The constant sum in every row, column and diagonal is called the **magic constant or magic sum**, M . The magic constant of a normal magic square depends only on n and has the following value:

$$M = n(n^2+1)/2$$

For normal magic squares of order $n = 3, 4, 5, \dots$, the magic constants are: 15, 34, 65, 111, 175, 260, ...

In this post, we will discuss how programmatically we can generate a magic square of size n . Before we go further, consider the below examples:

Magic Square of size 3

```
2  7  6
9  5  1
4  3  8
```

Sum in each row & each column = $3*(3^2+1)/2 = 15$

Magic Square of size 5

```
9  3  22  16  15
2  21  20  14  8
25 19  13  7  1
18 12  6  5  24
11 10  4  23  17
```

Sum in each row & each column = $5*(5^2+1)/2 = 65$

Magic Square of size 7

```
20 12  4  45  37  29  28
11  3  44  36  35  27  19
 2  43  42  34  26  18  10
49 41  33  25  17  9  1
40 32  24  16  8  7  48
31 23  15  14  6  47  39
22 21  13  5  46  38  30
```

Sum in each row & each column = $7*(7^2+1)/2 = 175$

Did you find any pattern in which the numbers are stored?

In any magic square, the first number i.e. 1 is stored at position $(n/2, n-1)$. Let this position be (i,j) . The next number is stored at position $(i-1, j+1)$ where we can consider each row & column as circular array i.e. they wrap around.

Three conditions hold:

1. The position of next number is calculated by decrementing row number of previous number by 1, and incrementing the column number of previous number by 1. At any time, if the calculated row position becomes -1, it will wrap around to $n-1$. Similarly, if the calculated column position becomes n , it will wrap around to 0.
2. If the magic square already contains a number at the calculated position, calculated column position will be decremented by 2, and calculated row position will be incremented by 1.
3. If the calculated row position is -1 & calculated column position is n , the new position would be: $(0, n-2)$.

Example:

Magic Square of size 3

```
-----
2 7 6
9 5 1
4 3 8
```

Steps:

1. position of number 1 = $(3/2, 3-1) = (1, 2)$
2. position of number 2 = $(1-1, 2+1) = (0, 0)$
3. position of number 3 = $(0-1, 0+1) = (3-1, 1) = (2, 1)$
4. position of number 4 = $(2-1, 1+1) = (1, 2)$
Since, at this position, 1 is there. So, apply condition 2.
new position= $(1+1, 2-2)=(2, 0)$
5. position of number 5= $(2-1, 0+1)=(1,1)$
6. position of number 6= $(1-1, 1+1)=(0,2)$
7. position of number 7 = $(0-1, 2+1) = (-1,3)$ // this is tricky, see condition 3
new position = $(0, 3-2) = (0,1)$
8. position of number 8= $(0-1, 1+1)=(-1,2)=(2,2)$ //wrap around
9. position of number 9= $(2-1, 2+1)=(1,3)=(1,0)$ //wrap around

Based on the above approach, following is the working code:

```
#include<stdio.h>
#include<string.h>

// A function to generate odd sized magic squares
void generateSquare(int n)
{
    int magicSquare[n][n];

    // set all slots as 0
    memset(magicSquare, 0, sizeof(magicSquare));

    // Initialize position for 1
    int i = n/2;
    int j = n-1;

    // One by one put all values in magic square
    for (int num=1; num <= n*n; )
    {
        if (i== -1 && j==n) //3rd condition
        {
            j = n-2;
            i = 0;
        }
        magicSquare[i][j] = num;
        num++;
        j--;
        if (j == -1)
            i++;
        else
            i--;
    }
}
```

```

else
{
    //1st condition helper if next number goes to out of square's right side
    if (j == n)
        j = 0;
    //1st condition helper if next number is goes to out of square's upper side
    if (i < 0)
        i=n-1;
}
if (magicSquare[i][j]) //2nd condition
{
    j -= 2;
    i++;
    continue;
}
else
    magicSquare[i][j] = num++; //set number

    j++; i--; //1st condition
}

// print magic square
printf("The Magic Square for n=%d:\nSum of each row or column %d:\n\n",
       n, n*(n*n+1)/2);
for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
        printf("%3d ", magicSquare[i][j]);
    printf("\n");
}
}

// Driver program to test above function
int main()
{
    int n = 7; // Works only when n is odd
    generateSquare (n);
    return 0;
}

```

[Run on IDE](#)

Output:

The Magic Square for n=7:
Sum of each row or column 175:

20	12	4	45	37	29	28
11	3	44	36	35	27	19
2	43	42	34	26	18	10
49	41	33	25	17	9	1
40	32	24	16	8	7	48
31	23	15	14	6	47	39
22	21	13	5	46	38	30

NOTE: This approach works only for odd values of n.

References:

http://en.wikipedia.org/wiki/Magic_square

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [Matrix](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

3.6

Average Difficulty : **3.6/5.0**
Based on 31 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Sieve of Eratosthenes

Given a number n , print all primes smaller than or equal to n . It is also given that n is a small number.

For example, if n is 10, the output should be "2, 3, 5, 7". If n is 20, the output should be "2, 3, 5, 7, 11, 13, 17, 19".

The sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than 10 million or so (Ref [Wiki](#)).

We strongly recommend that you click here and practice it, before moving on to the solution.

Following is the algorithm to find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

1. Create a list of consecutive integers from 2 to n : (2, 3, 4, ..., n).
2. Initially, let p equal 2, the first prime number.
3. Starting from p , count up in increments of p and mark each of these numbers greater than p itself in the list.
These numbers will be $2p$, $3p$, $4p$, etc.; note that some of them may have already been marked.
4. Find the first number greater than p in the list that is not marked. If there was no such number, stop.
Otherwise, let p now equal this number (which is the next prime), and repeat from step 3.

When the algorithm terminates, all the numbers in the list that are not marked are prime.

Explanation with Example:

Let us take an example when $n = 50$. So we need to print all print numbers smaller than or equal to 50.

We create a list of all numbers from 2 to 50.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

According to the algorithm we will mark all the numbers which are divisible by 2.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

We move to our next unmarked number 5 and mark all multiples of 5.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

We continue this process and our final table will look like below:

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

So the prime numbers are the unmarked ones: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

Thanks to **Krishan Kumar** for providing above explanation.

Implementation:

Following is C++ implementation of the above algorithm. In the following implementation, a boolean array arr[] of size n is used to mark multiples of prime numbers.

```
// C++ program to print all primes smaller than or equal to
// n using Sieve of Eratosthenes
#include <bits/stdc++.h>
using namespace std;

void SieveOfEratosthenes(int n)
{
    // Create a boolean array "prime[0..n]" and initialize
    // all entries it as true. A value in prime[i] will
    // finally be false if i is Not a prime, else true.
    bool prime[n+1];
    memset(prime, true, sizeof(prime));

    for (int p=2; p*p<=n; p++)
    {
        // If prime[p] is not changed, then it is a prime
        if (prime[p] == true)
        {
            // Update all multiples of p
            for (int i=p*p; i<=n; i += p)
                prime[i] = false;
        }
    }

    // Print all prime numbers
    for (int p=2; p<=n; p++)
        if (prime[p])
            cout << p << " ";
}
```

// Driver Program to test above function

```

int main()
{
    int n = 30;
    cout << "Following are the prime numbers smaller "
        << " than or equal to " << n << endl;
    SieveOfEratosthenes(n);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to print all primes smaller than or equal to
// n using Sieve of Eratosthenes

class SieveOfEratosthenes
{
    void sieveOfEratosthenes(int n)
    {
        // Create a boolean array "prime[0..n]" and initialize
        // all entries it as true. A value in prime[i] will
        // finally be false if i is Not a prime, else true.
        boolean prime[] = new boolean[n+1];
        for(int i=0;i<n;i++)
            prime[i] = true;

        for(int p = 2; p*p <=n; p++)
        {
            // If prime[p] is not changed, then it is a prime
            if(prime[p] == true)
            {
                // Update all multiples of p
                for(int i = p*2; i <= n; i += p)
                    prime[i] = false;
            }
        }

        // Print all prime numbers
        for(int i = 2; i <= n; i++)
        {
            if(prime[i] == true)
                System.out.print(i + " ");
        }
    }

    // Driver Program to test above function
    public static void main(String args[])
    {
        int n = 30;
        System.out.print("Following are the prime numbers ");
        System.out.println("smaller than or equal to " + n);
        SieveOfEratosthenes g = new SieveOfEratosthenes();
        g.sieveOfEratosthenes(n);
    }
}

// This code has been contributed by Amit Khandelwal.

```

[Run on IDE](#)

Python

```

# Python program to print all primes smaller than or equal to
# n using Sieve of Eratosthenes

def SieveOfEratosthenes(n):

```

```
# Create a boolean array "prime[0..n]" and initialize
# all entries it as true. A value in prime[i] will
# finally be false if i is Not a prime, else true.
prime = [True for i in range(n+1)]
p=2
while(p * p <= n):

    # If prime[p] is not changed, then it is a prime
    if (prime[p] == True):

        # Update all multiples of p
        for i in range(p * 2, n+1, p):
            prime[i] = False
        p+=1
lis = []

# Print all prime numbers
for p in range(2, n):
    if prime[p]:
        print p,

# driver program
if __name__=='__main__':
    n = 30
    print "Following are the prime numbers smaller",
    print "than or equal to", n
    SieveOfEratosthenes(n)
```

[Run on IDE](#)

Output:

```
Following are the prime numbers below 30
2 3 5 7 11 13 17 19 23 29
```

You may also like to see [Segmented Sieve](#).

References:

http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

This article is compiled by **Abhinav Priyadarshi** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Mathematical MathematicalAlgo prime-number sieve

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 52 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Find day of the week for a given date

Write a function that calculates the day of the week for any particular date in the past or future. A typical application is to calculate the day of the week on which someone was born or some other special event occurred.

Following is a simple C function suggested by [Sakamoto, Lachman, Keith and Craver](#) to calculate day. The following function returns 0 for Sunday, 1 for Monday, etc.

```
/* A program to find day of a given date */
#include<stdio.h>

int dayofweek(int d, int m, int y)
{
    static int t[] = { 0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4 };
    y -= m < 3;
    return ( y + y/4 - y/100 + y/400 + t[m-1] + d ) % 7;
}

/* Driver function to test above function */
int main()
{
    int day = dayofweek(30, 8, 2010);
    printf ("%d", day);

    return 0;
}
```

[Run on IDE](#)

Output: 1 (Monday)

See [this](#) for explanation of the above function.

References:

http://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week

This article is compiled by **Dheeraj Jain** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

Mathematical MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4.7

Average Difficulty : 4.7/5.0
Based on 18 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

DFA based division

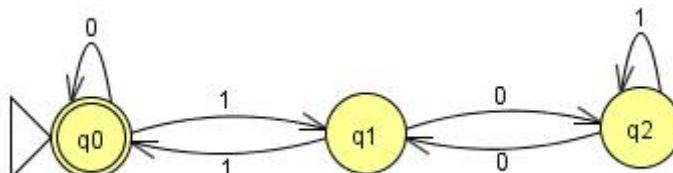
Deterministic Finite Automaton (DFA) can be used to check whether a number “num” is divisible by “k” or not. If the number is not divisible, remainder can also be obtained using DFA.

We consider the binary representation of ‘num’ and build a DFA with k states. The DFA has transition function for both 0 and 1. Once the DFA is built, we process ‘num’ over the DFA to get remainder.

Let us walk through an example. Suppose we want to check whether a given number ‘num’ is divisible by 3 or not. Any number can be written in the form: $\text{num} = 3*a + b$ where ‘a’ is the quotient and ‘b’ is the remainder.

For 3, there can be 3 states in DFA, each corresponding to remainder 0, 1 and 2. And each state can have two transitions corresponding 0 and 1 (considering the binary representation of given ‘num’).

The transition function $F(p, x) = q$ tells that on reading alphabet x, we move from state p to state q. Let us name the states as 0, 1 and 2. The initial state will always be 0. The final state indicates the remainder. If the final state is 0, the number is divisible.



In the above diagram, double circled state is final state.

1. When we are at state 0 and read 0, we remain at state 0.
2. When we are at state 0 and read 1, we move to state 1, why? The number so formed(1) in decimal gives remainder 1.
3. When we are at state 1 and read 0, we move to state 2, why? The number so formed(10) in decimal gives remainder 2.
4. When we are at state 1 and read 1, we move to state 0, why? The number so formed(11) in decimal gives remainder 0.
5. When we are at state 2 and read 0, we move to state 1, why? The number so formed(100) in decimal gives remainder 1.
6. When we are at state 2 and read 1, we remain at state 2, why? The number so formed(101) in decimal gives remainder 2.

The transition table looks like following:

state	0	1
q0	q0	q1
q1	q0	q2
q2	q2	q1

0	0	1
1	2	0
2	1	2

Let us check whether 6 is divisible by 3?

Binary representation of 6 is 110

state = 0

1. state=0, we read 1, new state=1
2. state=1, we read 1, new state=0
3. state=0, we read 0, new state=0

Since the final state is 0, the number is divisible by 3.

Let us take another example number as 4

state=0

1. state=0, we read 1, new state=1
2. state=1, we read 0, new state=2
3. state=2, we read 0, new state=1

Since, the final state is not 0, the number is not divisible by 3. The remainder is 1.

Note that the final state gives the remainder.

We can extend the above solution for any value of k. For a value k, the states would be 0, 1, ..., k-1. How to calculate the transition if the decimal equivalent of the binary bits seen so far, crosses the range k? If we are at state p, we have read p (in decimal). Now we read 0, new read number becomes 2^p . If we read 1, new read number becomes $2^p + 1$. The new state can be obtained by subtracting k from these values (2^p or $2^p + 1$) where $0 \leq p < k$. Based on the above approach, following is the working code:

```
[sourcecode language="C" highlight="35,36-50"]
#include <stdio.h> #include <stdlib.h> // Function to build DFA for divisor k void preprocess(int k, int Table[][2]) { int trans0, trans1; // The following loop calculates the two transitions for each state, // starting from state 0 for (int state=0; state<k; ++state) { // Calculate next state for bit 0 trans0 = state<<1; Table[state][0] = (trans0 < k)? trans0: trans0-k; // Calculate next state for bit 1 trans1 = (state<<1) + 1; Table[state][1] = (trans1 < k)? trans1: trans1-k; } } // A recursive utility function that takes a 'num' and DFA (transition // table) as input and process 'num' bit by bit over DFA void isDivisibleUtil(int num, int* state, int Table[][2]) { // process "num" bit by bit from MSB to LSB if (num != 0) { isDivisibleUtil(num>>1, state, Table); *state = Table[*state][num&1]; } } // The main function that divides 'num' by k and returns the remainder int isDivisible (int num, int k) { // Allocate memory for transition table. The table will have k*2 entries int (*Table)[2] = (int (*)[2])malloc(k*sizeof(*Table)); // Fill the transition table preprocess(k, Table); // Process 'num' over DFA and get the remainder int state = 0; isDivisibleUtil(num, &state, Table); // Note that the final value of state is the remainder return state; } // Driver program to test above functions int main() { int num = 47; // Number to be divided int k = 5; // Divisor int remainder = isDivisible (num, k); if (remainder == 0) printf("Divisible\n"); else printf("Not Divisible: Remainder is %d\n", remainder); return 0; }
```

[/sourcecode] Output:

Not Divisible: Remainder is 2

DFA based division can be useful if we have a binary stream as input and we want to check for divisibility of the decimal value of stream at any time.

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Ali Tile BY **LOLA GROUP**

LOLA CERAMICS (M) SDN. BHD.
ADDRESS: 38, Jalan PJS 343, Sunway Damansara, 4719 Petaling Jaya, Malaysia
TEL: +603-9056 2910 WEBSITE: www.alatile.com

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alatile.com

GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4.2

Average Difficulty : **4.2/5.0**
Based on **12** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Generate integer from 1 to 7 with equal probability

Given a function `foo()` that returns integers from 1 to 5 with equal probability, write a function that returns integers from 1 to 7 with equal probability using `foo()` only. Minimize the number of calls to `foo()` method. Also, use of any other library function is not allowed and no floating point arithmetic allowed.

Solution:

We know `foo()` returns integers from 1 to 5. How we can ensure that integers from 1 to 7 occur with equal probability?

If we somehow generate integers from 1 to a-multiple-of-7 (like 7, 14, 21, ...) with equal probability, we can use modulo division by 7 followed by adding 1 to get the numbers from 1 to 7 with equal probability.

We can generate from 1 to 21 with equal probability using the following expression.

```
5*foo() + foo() -5
```

Let us see how above expression can be used.

1. For each value of first `foo()`, there can be 5 possible combinations for values of second `foo()`. So, there are total 25 combinations possible.
2. The range of values returned by the above equation is 1 to 25, each integer occurring exactly once.
3. If the value of the equation comes out to be less than 22, return modulo division by 7 followed by adding 1. Else, again call the method recursively. The probability of returning each integer thus becomes 1/7.

The below program shows that the expression returns each integer from 1 to 25 exactly once.

```
#include <stdio.h>

int main()
{
    int first, second;
    for ( first=1; first<=5; ++first )
        for ( second=1; second<=5; ++second )
            printf ("%d \n", 5*first + second - 5);
    return 0;
}
```

[Run on IDE](#)

Output:

```
1
2
.
.
```

24
25

The below program depicts how we can use foo() to return 1 to 7 with equal probability.

```
#include <stdio.h>

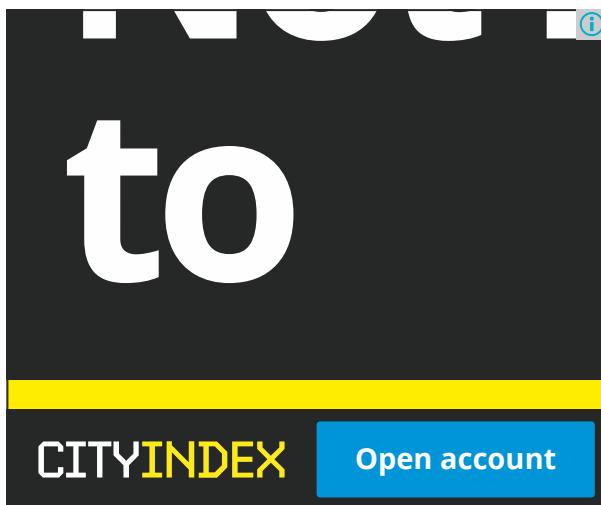
int foo() // given method that returns 1 to 5 with equal probability
{
    // some code here
}

int my_rand() // returns 1 to 7 with equal probability
{
    int i;
    i = 5*foo() + foo() - 5;
    if (i < 22)
        return i%7 + 1;
    return my_rand();
}

int main()
{
    printf ("%d ", my_rand());
    return 0;
}
```

[Run on IDE](#)

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Randomized](#) [MathematicalAlgo](#)

Related Posts:

- Program to generate CAPTCHA and verify user
- Find an index of maximum occurring element with equal probability
- Implement rand12() using rand6() in one line

- Implement rand3() using rand2()
- Generate 0 and 1 with 25% and 75% probability
- Randomized Algorithms | Set 0 (Mathematical Background)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)
- Primality Test | Set 2 (Fermat Method)

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **17** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Given a number, find the next smallest palindrome

Given a number, find the next smallest palindrome larger than this number. For example, if the input number is “2 3 5 4 5”, the output should be “2 3 6 3 2”. And if the input number is “9 9 9”, the output should be “1 0 0 1”.

The input is assumed to be an array. Every entry in array represents a digit in input number. Let the array be ‘num[]’ and size of array be ‘n’

There can be three different types of inputs that need to be handled separately.

- 1) The input number is palindrome and has all 9s. For example “9 9 9”. Output should be “1 0 0 1”
- 2) The input number is not palindrome. For example “1 2 3 4”. Output should be “1 3 3 1”
- 3) The input number is palindrome and doesn’t have all 9s. For example “1 2 2 1”. Output should be “1 3 3 1”.

We strongly recommend that you click here and practice it, before moving on to the solution.

Solution for input type 1 is easy. The output contains $n + 1$ digits where the corner digits are 1, and all digits between corner digits are 0.

Now let us first talk about input type 2 and 3. How to convert a given number to a greater palindrome? To understand the solution, let us first define the following two terms:

Left Side: The left half of given number. Left side of “1 2 3 4 5 6” is “1 2 3” and left side of “1 2 3 4 5” is “1 2”

Right Side: The right half of given number. Right side of “1 2 3 4 5 6” is “4 5 6” and right side of “1 2 3 4 5” is “4 5”

To convert to palindrome, we can either take the mirror of its left side or take mirror of its right side. However, if we take the mirror of the right side, then the palindrome so formed is not guaranteed to be next larger palindrome. So, we must take the mirror of left side and copy it to right side. But there are some cases that must be handled in different ways. See the following steps.

We will start with two indices i and j. i pointing to the two middle elements (or pointing to two elements around the middle element in case of n being odd). We one by one move i and j away from each other.

Step 1. Initially, ignore the part of left side which is same as the corresponding part of right side. For example, if the number is “8 3 4 2 2 4 6 9”, we ignore the middle four elements. i now points to element 3 and j now points to element 6.

Step 2. After step 1, following cases arise:

Case 1: Indices i & j cross the boundary.

This case occurs when the input number is palindrome. In this case, we just add 1 to the middle digit (or digits in case n is even) propagate the carry towards MSB digit of left side and simultaneously copy mirror of the left side to the right side.

For example, if the given number is “1 2 9 2 1”, we increment 9 to 10 and propagate the carry. So the number becomes “1 3 0 3 1”

Case 2: There are digits left between left side and right side which are not same. So, we just mirror the left side to the right side & try to minimize the number formed to guarantee the next smallest palindrome.

In this case, there can be **two sub-cases**.

2.1) Copying the left side to the right side is sufficient, we don't need to increment any digits and the result is just mirror of left side. Following are some examples of this sub-case.

Next palindrome for “7 8 3 3 2 2” is “7 8 3 3 8 7”

Next palindrome for “1 2 5 3 2 2” is “1 2 5 5 2 1”

Next palindrome for “1 4 5 8 7 6 7 8 3 2 2” is “1 4 5 8 7 6 7 8 5 4 1”

How do we check for this sub-case? All we need to check is the digit just after the ignored part in step 1. This digit is highlighted in above examples. If this digit is greater than the corresponding digit in right side digit, then copying the left side to the right side is sufficient and we don't need to do anything else.

2.2) Copying the left side to the right side is NOT sufficient. This happens when the above defined digit of left side is smaller. Following are some examples of this case.

Next palindrome for “7 1 3 3 2 2” is “7 1 4 4 1 7”

Next palindrome for “1 2 3 4 6 2 8” is “1 2 3 5 3 2 1”

Next palindrome for “9 4 1 8 7 9 7 8 3 2 2” is “9 4 1 8 8 0 8 8 1 4 9”

We handle this subcase like Case 1. We just add 1 to the middle digit (or digits in case n is even) propagate the carry towards MSB digit of left side and simultaneously copy mirror of the left side to the right side.

```
#include <stdio.h>

// A utility function to print an array
void printArray (int arr[], int n);

// A utility function to check if num has all 9s
int AreAll9s (int num[], int n );

// Returns next palindrome of a given number num[].
// This function is for input type 2 and 3
void generateNextPalindromeUtil (int num[], int n )
{
    // find the index of mid digit
    int mid = n/2;

    // A bool variable to check if copy of left side to right is sufficient or not
    bool leftsmaller = false;

    // end of left side is always 'mid -1'
    int i = mid - 1;

    // Begining of right side depends if n is odd or even
    int j = (n % 2)? mid + 1 : mid;

    // Initially, ignore the middle same digits
    while (i >= 0 && num[i] == num[j])
        i--,j++;

    // Find if the middle digit(s) need to be incremented or not (or copying left
    // side is not sufficient)
    if ( i < 0 || num[i] < num[j])
        leftsmaller = true;
```

```

// Copy the mirror of left to tight
while (i >= 0)
{
    num[j] = num[i];
    j++;
    i--;
}

// Handle the case where middle digit(s) must be incremented.
// This part of code is for CASE 1 and CASE 2.2
if (leftsmaller == true)
{
    int carry = 1;
    i = mid - 1;

    // If there are odd digits, then increment
    // the middle digit and store the carry
    if (n%2 == 1)
    {
        num[mid] += carry;
        carry = num[mid] / 10;
        num[mid] %= 10;
        j = mid + 1;
    }
    else
        j = mid;

    // Add 1 to the rightmost digit of the left side, propagate the carry
    // towards MSB digit and simultaneously copying mirror of the left side
    // to the right side.
    while (i >= 0)
    {
        num[i] += carry;
        carry = num[i] / 10;
        num[i] %= 10;
        num[j++] = num[i--]; // copy mirror to right
    }
}

// The function that prints next palindrome of a given number num[]
// with n digits.
void generateNextPalindrome( int num[], int n )
{
    int i;

    printf("\nNext palindrome is:\n");

    // Input type 1: All the digits are 9, simply o/p 1
    // followed by n-1 0's followed by 1.
    if( AreAll9s( num, n ) )
    {
        printf( "1 " );
        for( i = 1; i < n; i++ )
            printf( "0 " );
        printf( "1" );
    }

    // Input type 2 and 3
    else
    {
        generateNextPalindromeUtil ( num, n );

        // print the result
        printArray (num, n);
    }
}

// A utility function to check if num has all 9s
int AreAll9s( int* num, int n )
{
    int i;
    for( i = 0; i < n; ++i )
        if( num[i] != 9 )

```

```

        return 0;
    return 1;
}

/* Utility that prints out an array on a line */
void printArray(int arr[], int n)
{
    int i;
    for (i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver Program to test above function
int main()
{
    int num[] = {9, 4, 1, 8, 7, 9, 7, 8, 3, 2, 2};
    int n = sizeof (num)/ sizeof(num[0]);
    generateNextPalindrome( num, n );
    return 0;
}

```

[Run on IDE](#)

Output:

```

Next palindrome is:
9 4 1 8 8 0 8 8 1 4 9

```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 45 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Make a fair coin from a biased coin

You are given a function `foo()` that represents a biased coin. When `foo()` is called, it returns 0 with 60% probability, and 1 with 40% probability. Write a new function that returns 0 and 1 with 50% probability each. Your function should use only `foo()`, no other library method.

Solution:

We know `foo()` returns 0 with 60% probability. How can we ensure that 0 and 1 are returned with 50% probability?

The solution is similar to [this](#) post. If we can somehow get two cases with equal probability, then we are done. We call `foo()` two times. Both calls will return 0 with 60% probability. So the two pairs (0, 1) and (1, 0) will be generated with equal probability from two calls of `foo()`. Let us see how.

(0, 1): The probability to get 0 followed by 1 from two calls of `foo()` = $0.6 * 0.4 = 0.24$

(1, 0): The probability to get 1 followed by 0 from two calls of `foo()` = $0.4 * 0.6 = 0.24$

So the two cases appear with equal probability. The idea is to return consider only the above two cases, return 0 in one case, return 1 in other case. For other cases [(0, 0) and (1, 1)], recur until you end up in any of the above two cases.

The below program depicts how we can use `foo()` to return 0 and 1 with equal probability.

```
#include <stdio.h>

int foo() // given method that returns 0 with 60% probability and 1 with 40%
{
    // some code here
}

// returns both 0 and 1 with 50% probability
int my_fun()
{
    int val1 = foo();
    int val2 = foo();
    if (val1 == 0 && val2 == 1)
        return 0; // Will reach here with 0.24 probability
    if (val1 == 1 && val2 == 0)
        return 1; // Will reach here with 0.24 probability
    return my_fun(); // will reach here with (1 - 0.24 - 0.24) probability
}

int main()
{
    printf ("%d ", my_fun());
    return 0;
}
```

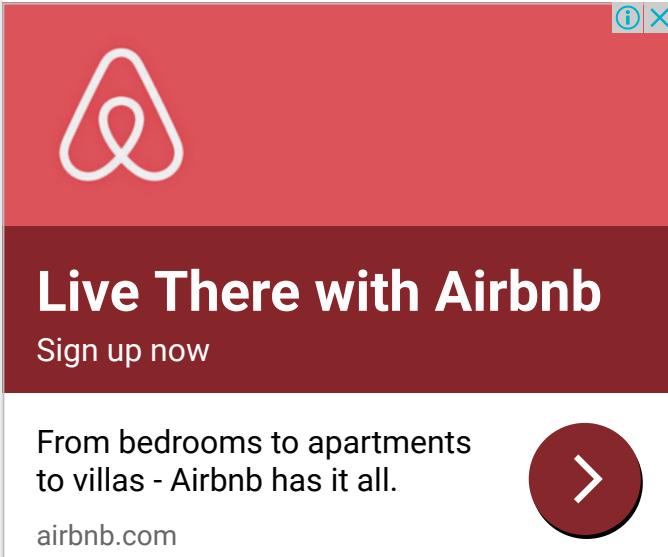
[Run on IDE](#)

References:

http://en.wikipedia.org/wiki/Fair_coin#Fair_results_from_a_biased_coin

This article is compiled by **Shashank Sinha** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.



GATE CS Corner Company Wise Coding Practice

Randomized MathematicalAlgo

Related Posts:

- Program to generate CAPTCHA and verify user
- Find an index of maximum occurring element with equal probability
- Implement rand12() using rand6() in one line
- Implement rand3() using rand2()
- Generate 0 and 1 with 25% and 75% probability
- Randomized Algorithms | Set 0 (Mathematical Background)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)
- Primality Test | Set 2 (Fermat Method)

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 11 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Check divisibility by 7

Given a number, check if it is divisible by 7. You are not allowed to use modulo operator, floating point arithmetic is also not allowed.

A simple method is repeated subtraction. Following is another interesting method.

Divisibility by 7 can be checked by a recursive method. A number of the form $10a + b$ is divisible by 7 if and only if $a - 2b$ is divisible by 7. In other words, subtract twice the last digit from the number formed by the remaining digits. Continue to do this until a small number.

Example: the number 371: $37 - (2 \times 1) = 37 - 2 = 35$; $3 - (2 \times 5) = 3 - 10 = -7$; thus, since -7 is divisible by 7, 371 is divisible by 7.

Following is C implementation of the above method

```
// A Program to check whether a number is divisible by 7
#include <stdio.h>

int isDivisibleBy7( int num )
{
    // If number is negative, make it positive
    if( num < 0 )
        return isDivisibleBy7( -num );

    // Base cases
    if( num == 0 || num == 7 )
        return 1;
    if( num < 10 )
        return 0;

    // Recur for ( num / 10 - 2 * num % 10 )
    return isDivisibleBy7( num / 10 - 2 * ( num - num / 10 * 10 ) );
}

// Driver program to test above function
int main()
{
    int num = 616;
    if( isDivisibleBy7(num) )
        printf( "Divisible" );
    else
        printf( "Not Divisible" );
    return 0;
}
```

[Run on IDE](#)

Output:

Divisible

How does this work? Let 'b' be the last digit of a number 'n' and let 'a' be the number we get when we split off 'b'. The representation of the number may also be multiplied by any number relatively prime to the divisor without changing its divisibility. After observing that 7 divides 21, we can perform the following:

$$10.a + b$$

after multiplying by 2, this becomes

$$20.a + 2.b$$

and then

$$21.a - a + 2.b$$

Eliminating the multiple of 21 gives

$$-a + 2b$$

and multiplying by -1 gives

$$a - 2b$$

There are other interesting methods to check divisibility by 7 and other numbers. See following Wiki page for details.

References:

http://en.wikipedia.org/wiki/Divisibility_rule

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

The advertisement features a collage of images: a modern building with a sign for 'LOLA CERAMICS (M) SDN. BHD.', a group of six men standing outdoors, and a close-up of a tile. To the right, the 'Alitile BY LOLA GROUP' logo is displayed with a '①' icon. Below the images, the text 'China huge tile factory' is written in orange. At the bottom left, it says 'Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia'. A circular orange button with a white arrow points to the right. The website 'alitile.com' is listed at the bottom.

GATE CS Corner Company Wise Coding Practice

[Mathematical](#)[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.5

Average Difficulty : **2.5/5.0**
Based on 7 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find the largest multiple of 3

Given an array of non-negative integers. Find the largest multiple of 3 that can be formed from array elements.

For example, if the input array is {8, 1, 9}, the output should be "9 8 1", and if the input array is {8, 1, 7, 6, 0}, output should be "8 7 6 0".

Method 1 (Brute Force)

The simple & straight forward approach is to generate all the combinations of the elements and keep track of the largest number formed which is divisible by 3.

Time Complexity: $O(n \times 2^n)$. There will be 2^n combinations of array elements. To compare each combination with the largest number so far may take $O(n)$ time.

Auxiliary Space: $O(n)$ // to avoid integer overflow, the largest number is assumed to be stored in the form of array.

Method 2 (Tricky)

This problem can be solved efficiently with the help of $O(n)$ extra space. This method is based on the following facts about numbers which are multiple of 3.

1) A number is multiple of 3 if and only if the sum of digits of number is multiple of 3. For example, let us consider 8760, it is a multiple of 3 because sum of digits is $8 + 7 + 6 + 0 = 21$, which is a multiple of 3.

2) If a number is multiple of 3, then all permutations of it are also multiple of 3. For example, since 6078 is a multiple of 3, the numbers 8760, 7608, 7068, are also multiples of 3.

3) We get the same remainder when we divide the number and sum of digits of the number. For example, if divide number 151 and sum of its digits 7, by 3, we get the same remainder 1.

What is the idea behind above facts?

The value of $10\%3$ and $100\%3$ is 1. The same is true for all the higher powers of 10, because 3 divides 9, 99, 999, ... etc.

Let us consider a 3 digit number n to prove above facts. Let the first, second and third digits of n be 'a', 'b' and 'c' respectively. n can be written as

$$n = 100.a + 10.b + c$$

Since $(10^x)\%3$ is 1 for any x, the above expression gives the same remainder as following expression

$$1.a + 1.b + c$$

So the remainder obtained by sum of digits and 'n' is same.

Following is a solution based on the above observation.

1. Sort the array in non-decreasing order.

2. Take three queues. One for storing elements which on dividing by 3 gives remainder as 0. The second queue stores digits which on dividing by 3 gives remainder as 1. The third queue stores digits which on dividing by 3 gives remainder as 2. Call them as queue0, queue1 and queue2

3. Find the sum of all the digits.

4. Three cases arise:

.....**4.1** The sum of digits is divisible by 3. Dequeue all the digits from the three queues. Sort them in non-increasing order. Output the array.

.....**4.2** The sum of digits produces remainder 1 when divided by 3.

Remove one item from queue1. If queue1 is empty, remove two items from queue2. If queue2 contains less than two items, the number is not possible.

.....**4.3** The sum of digits produces remainder 2 when divided by 3.

Remove one item from queue2. If queue2 is empty, remove two items from queue1. If queue1 contains less than two items, the number is not possible.

5. Finally empty all the queues into an auxiliary array. Sort the auxiliary array in non-increasing order. Output the auxiliary array.

Based on the above, below is C implementation:

The below code works only if the input arrays has numbers from 0 to 9. It can be easily extended for any positive integer array. We just have to modify the part where we sort the array in decreasing order, at the end of code.

```
/* A program to find the largest multiple of 3 from an array of elements */
#include <stdio.h>
#include <stdlib.h>

// A queue node
typedef struct Queue
{
    int front;
    int rear;
    int capacity;
    int* array;
} Queue;

// A utility function to create a queue with given capacity
Queue* createQueue( int capacity )
{
    Queue* queue = (Queue *) malloc (sizeof(Queue));
    queue->capacity = capacity;
    queue->front = queue->rear = -1;
    queue->array = (int *) malloc (queue->capacity * sizeof(int));
    return queue;
}

// A utility function to check if queue is empty
int isEmpty (Queue* queue)
{
    return queue->front == -1;
}

// A function to add an item to queue
void Enqueue (Queue* queue, int item)
{
    queue->array[ ++queue->rear ] = item;
    if ( isEmpty(queue) )
```

```

        ++queue->front;
    }

// A function to remove an item from queue
int Dequeue (Queue* queue)
{
    int item = queue->array[ queue->front ];
    if( queue->front == queue->rear )
        queue->front = queue->rear = -1;
    else
        queue->front++;

    return item;
}

// A utility function to print array contents
void printArr (int* arr, int size)
{
    int i;
    for (i = 0; i < size; ++i)
        printf ("%d ", arr[i]);
}

/* Following two functions are needed for library function qsort().
Refer following link for help of qsort()
http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compareAsc( const void* a, const void* b )
{
    return *(int*)a > *(int*)b;
}
int compareDesc( const void* a, const void* b )
{
    return *(int*)a < *(int*)b;
}

// This function puts all elements of 3 queues in the auxiliary array
void populateAux (int* aux, Queue* queue0, Queue* queue1,
                  Queue* queue2, int* top )
{
    // Put all items of first queue in aux[]
    while ( !isEmpty(queue0) )
        aux[ (*top)++ ] = Dequeue( queue0 );

    // Put all items of second queue in aux[]
    while ( !isEmpty(queue1) )
        aux[ (*top)++ ] = Dequeue( queue1 );

    // Put all items of third queue in aux[]
    while ( !isEmpty(queue2) )
        aux[ (*top)++ ] = Dequeue( queue2 );
}

// The main function that finds the largest possible multiple of
// 3 that can be formed by arr[] elements
int findMaxMultipleOf3( int* arr, int size )
{
    // Step 1: sort the array in non-decreasing order
    qsort( arr, size, sizeof( int ), compareAsc );

    // Create 3 queues to store numbers with remainder 0, 1
    // and 2 respectively
    Queue* queue0 = createQueue( size );
    Queue* queue1 = createQueue( size );
    Queue* queue2 = createQueue( size );

    // Step 2 and 3 get the sum of numbers and place them in
    // corresponding queues
    int i, sum;
    for ( i = 0, sum = 0; i < size; ++i )
    {
        sum += arr[i];
        if ( (arr[i] % 3) == 0 )
            Enqueue( queue0, arr[i] );
        else if ( (arr[i] % 3) == 1 )
            Enqueue( queue1, arr[i] );
        else
            Enqueue( queue2, arr[i] );
    }
}

```

```

        Enqueue( queue1, arr[i] );
    else
        Enqueue( queue2, arr[i] );
}

// Step 4.2: The sum produces remainder 1
if ( (sum % 3) == 1 )
{
    // either remove one item from queue1
    if ( !isEmpty( queue1 ) )
        Dequeue( queue1 );

    // or remove two items from queue2
    else
    {
        if ( !isEmpty( queue2 ) )
            Dequeue( queue2 );
        else
            return 0;

        if ( !isEmpty( queue2 ) )
            Dequeue( queue2 );
        else
            return 0;
    }
}

// Step 4.3: The sum produces remainder 2
else if ((sum % 3) == 2)
{
    // either remove one item from queue2
    if ( !isEmpty( queue2 ) )
        Dequeue( queue2 );

    // or remove two items from queue1
    else
    {
        if ( !isEmpty( queue1 ) )
            Dequeue( queue1 );
        else
            return 0;

        if ( !isEmpty( queue1 ) )
            Dequeue( queue1 );
        else
            return 0;
    }
}

int aux[size], top = 0;

// Empty all the queues into an auxiliary array.
populateAux (aux, queue0, queue1, queue2, &top);

// sort the array in non-increasing order
qsort (aux, top, sizeof( int ), compareDesc);

// print the result
printArr (aux, top);

return top;
}

// Driver program to test above functions
int main()
{
    int arr[] = {8, 1, 7, 6, 0};
    int size = sizeof(arr)/sizeof(arr[0]);

    if (findMaxMultipleOf3( arr, size ) == 0)
        printf( "Not Possible" );

    return 0;
}

```

The above method can be optimized in following ways.

- 1) We can use Heap Sort or Merge Sort to make the time complexity $O(n\log n)$.
- 2) We can avoid extra space for queues. We know at most two items will be removed from the input array. So we can keep track of two items in two variables.
- 3) At the end, instead of sorting the array again in descending order, we can print the ascending sorted array in reverse order. While printing in reverse order, we can skip the two elements to be removed.

Time Complexity: $O(n\log n)$, assuming a $O(n\log n)$ algorithm is used for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Queue](#) [MathematicalAlgo](#)

Related Posts:

- Find maximum level sum in Binary Tree
- Implement a stack using single queue
- Minimum time required to rot all oranges
- How to efficiently implement k Queues in a single array?
- An Interesting Method to Generate Binary Numbers from 1 to n
- Iterative Method to find Height of Binary Tree
- Construct Complete Binary Tree from its Linked List Representation
- Find the first circular tour that visits all petrol pumps

[\(Login to Rate and Mark\)](#)**4.1**Average Difficulty : **4.1/5.0**
Based on **75** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Lexicographic rank of a string

Given a string, find its rank among all its permutations sorted lexicographically. For example, rank of “abc” is 1, rank of “acb” is 2, and rank of “cba” is 6.

We strongly recommend that you click here and practice it, before moving on to the solution.

For simplicity, let us assume that the string does not contain any duplicated characters.

One simple solution is to initialize rank as 1, generate all permutations in lexicographic order. After generating a permutation, check if the generated permutation is same as given string, if same, then return rank, if not, then increment the rank by 1. The time complexity of this solution will be exponential in worst case. Following is an efficient solution.

Let the given string be “STRING”. In the input string, ‘S’ is the first character. There are total 6 characters and 4 of them are smaller than ‘S’. So there can be $4 * 5!$ smaller strings where first character is smaller than ‘S’, like following

R X X X X X

I X X X X X

N X X X X X

G X X X X X

Now let us Fix S’ and find the smaller strings staring with ‘S’.

Repeat the same process for T, rank is $4*5! + 4*4! + \dots$

Now fix T and repeat the same process for R, rank is $4*5! + 4*4! + 3*3! + \dots$

Now fix R and repeat the same process for I, rank is $4*5! + 4*4! + 3*3! + 1*2! + \dots$

Now fix I and repeat the same process for N, rank is $4*5! + 4*4! + 3*3! + 1*2! + 1*1! + \dots$

Now fix N and repeat the same process for G, rank is $4*5! + 4*4! + 3*3! + 1*2! + 1*1! + 0*0!$

Rank = $4*5! + 4*4! + 3*3! + 1*2! + 1*1! + 0*0! = 597$

Since the value of rank starts from 1, the final rank = $1 + 597 = 598$

```
#include <stdio.h>
```

```
#include <string.h>

// A utility function to find factorial of n
int fact(int n)
{
    return (n <= 1)? 1 : n * fact(n-1);
}

// A utility function to count smaller characters on right
// of arr[low]
int findSmallerInRight(char* str, int low, int high)
{
    int countRight = 0, i;

    for (i = low+1; i <= high; ++i)
        if (str[i] < str[low])
            ++countRight;

    return countRight;
}

// A function to find rank of a string in all permutations
// of characters
int findRank (char* str)
{
    int len = strlen(str);
    int mul = fact(len);
    int rank = 1;
    int countRight;

    int i;
    for (i = 0; i < len; ++i)
    {
        mul /= len - i;

        // count number of chars smaller than str[i]
        // from str[i+1] to str[len-1]
        countRight = findSmallerInRight(str, i, len-1);

        rank += countRight * mul ;
    }

    return rank;
}

// Driver program to test above function
int main()
{
    char str[] = "string";
    printf ("%d", findRank(str));
    return 0;
}
```

[Run on IDE](#)

Output

598

The time complexity of the above solution is $O(n^2)$. We can reduce the time complexity to $O(n)$ by creating an auxiliary array of size 256. See following code.

```
// A O(n) solution for finding rank of string
#include <stdio.h>
#include <string.h>
#define MAX_CHAR 256

// A utility function to find factorial of n
int fact(int n)
{
    return (n <= 1)? 1 : n * fact(n-1);
```

```

}

// Construct a count array where value at every index
// contains count of smaller characters in whole string
void populateAndIncreaseCount (int* count, char* str)
{
    int i;

    for( i = 0; str[i]; ++i )
        ++count[ str[i] ];

    for( i = 1; i < 256; ++i )
        count[i] += count[i-1];
}

// Removes a character ch from count[] array
// constructed by populateAndIncreaseCount()
void updateCount (int* count, char ch)
{
    int i;
    for( i = ch; i < MAX_CHAR; ++i )
        --count[i];
}

// A function to find rank of a string in all permutations
// of characters
int findRank (char* str)
{
    int len = strlen(str);
    int mul = fact(len);
    int rank = 1, i;
    int count[MAX_CHAR] = {0}; // all elements of count[] are initialized with 0

    // Populate the count array such that count[i] contains count of
    // characters which are present in str and are smaller than i
    populateAndIncreaseCount( count, str );

    for (i = 0; i < len; ++i)
    {
        mul /= len - i;

        // count number of chars smaller than str[i]
        // from str[i+1] to str[len-1]
        rank += count[ str[i] - 1 ] * mul;

        // Reduce count of characters greater than str[i]
        updateCount (count, str[i]);
    }

    return rank;
}

// Driver program to test above function
int main()
{
    char str[] = "string";
    printf ("%d", findRank(str));
    return 0;
}

```

Run on IDE

The above programs don't work for duplicate characters. To make them work for duplicate characters, find all the characters that are smaller (include equal this time also), do the same as above but, this time divide the rank so formed by $p!$ where p is the count of occurrences of the repeating character.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

[Strings](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Convert to a string that is repetition of a substring of k length
- Minimum characters to be added at front to make string palindrome
- Count All Palindrome Sub-Strings in a String
- Check for Palindrome after every character replacement Query
- Group all occurrences of characters according to first appearance
- Count characters at same position as in English alphabets
- Find if an array of strings can be chained to form a circle | Set 2

(Login to Rate and Mark)

3.7

Average Difficulty : 3.7/5.0
Based on 49 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Print all permutations in sorted (lexicographic) order

Given a string, print all permutations of it in sorted order. For example, if the input string is “ABC”, then output should be “ABC, ACB, BAC, BCA, CAB, CBA”.

We have discussed a program to print all permutations in [this](#) post, but here we must print the permutations in increasing order.

Following are the steps to print the permutations lexicographic-ally

1. Sort the given string in non-decreasing order and print it. The first permutation is always the string sorted in non-decreasing order.
2. Start generating next higher permutation. Do it until next higher permutation is not possible. If we reach a permutation where all characters are sorted in non-increasing order, then that permutation is the last permutation.

Steps to generate the next higher permutation:

1. Take the previously printed permutation and find the rightmost character in it, which is smaller than its next character. Let us call this character as ‘first character’.
2. Now find the ceiling of the ‘first character’. Ceiling is the smallest character on right of ‘first character’, which is greater than ‘first character’. Let us call the ceil character as ‘second character’.
3. Swap the two characters found in above 2 steps.
4. Sort the substring (in non-decreasing order) after the original index of ‘first character’.

Let us consider the string “ABCDEF”. Let previously printed permutation be “DCFEBA”. The next permutation in sorted order should be “DEABC”. Let us understand above steps to find next permutation. The ‘first character’ will be ‘C’. The ‘second character’ will be ‘E’. After swapping these two, we get “DEFBCA”. The final step is to sort the substring after the character original index of ‘first character’. Finally, we get “DEABC”.

Following is C++ implementation of the algorithm.

```
// Program to print all permutations of a string in sorted order.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 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 two swap two characters a and b
void swap (char* a, char* b)
{
    char t = *a;
```

```

*a = *b;
*b = t;
}

// This function finds the index of the smallest character
// which is greater than 'first' and is present in str[l..h]
int findCeil (char str[], char first, int l, int h)
{
    // initialize index of ceiling element
    int ceilIndex = l;

    // Now iterate through rest of the elements and find
    // the smallest character greater than 'first'
    for (int i = l+1; i <= h; i++)
        if (str[i] > first && str[i] < str[ceilIndex])
            ceilIndex = i;

    return ceilIndex;
}

// Print all permutations of str in sorted order
void sortedPermutations ( char str[] )
{
    // Get size of string
    int size = strlen(str);

    // Sort the string in increasing order
    qsort( str, size, sizeof( str[0] ), compare );

    // Print permutations one by one
    bool isFinished = false;
    while ( ! isFinished )
    {
        // print this permutation
        printf ("%s \n", str);

        // Find the rightmost character which is smaller than its next
        // character. Let us call it 'first char'
        int i;
        for ( i = size - 2; i >= 0; --i )
            if (str[i] < str[i+1])
                break;

        // If there is no such character, all are sorted in decreasing order,
        // means we just printed the last permutation and we are done.
        if ( i == -1 )
            isFinished = true;
        else
        {
            // Find the ceil of 'first char' in right of first character.
            // Ceil of a character is the smallest character greater than it
            int ceilIndex = findCeil( str, str[i], i + 1, size - 1 );

            // Swap first and second characters
            swap( &str[i], &str[ceilIndex] );

            // Sort the string on right of 'first char'
            qsort( str + i + 1, size - i - 1, sizeof(str[0]), compare );
        }
    }
}

// Driver program to test above function
int main()
{
    char str[] = "ABCD";
    sortedPermutations( str );
    return 0;
}

```

Run on IDE

Output:

```
ABCD
ABDC
....
....
DCAB
DCBA
```

The upper bound on time complexity of the above program is $O(n^2 \times n!)$. We can optimize step 4 of the above algorithm for finding next permutation. Instead of sorting the subarray after the 'first character', we can reverse the subarray, because the subarray we get after swapping is always sorted in non-increasing order. This optimization makes the time complexity as $O(n \times n!)$. See following optimized code.

```
// An optimized version that uses reverse instead of sort for
// finding the next permutation

// A utility function to reverse a string str[l..h]
void reverse(char str[], int l, int h)
{
    while (l < h)
    {
        swap(&str[l], &str[h]);
        l++;
        h--;
    }
}

// Print all permutations of str in sorted order
void sortedPermutations (char str[])
{
    // Get size of string
    int size = strlen(str);

    // Sort the string in increasing order
    qsort( str, size, sizeof( str[0] ), compare );

    // Print permutations one by one
    bool isFinished = false;
    while ( ! isFinished )
    {
        // print this permutation
        printf ("%s \n", str);

        // Find the rightmost character which is smaller than its next
        // character. Let us call it 'first char'
        int i;
        for ( i = size - 2; i >= 0; --i )
            if (str[i] < str[i+1])
                break;

        // If there is no such character, all are sorted in decreasing order,
        // means we just printed the last permutation and we are done.
        if ( i == -1 )
            isFinished = true;
        else
        {
            // Find the ceil of 'first char' in right of first character.
            // Ceil of a character is the smallest character greater than it
            int ceilIndex = findCeil( str, str[i], i + 1, size - 1 );

            // Swap first and second characters
            swap( &str[i], &str[ceilIndex] );

            // reverse the string on right of 'first char'
            reverse( str, i + 1, size - 1 );
        }
    }
}
```

[Run on IDE](#)

The above programs print duplicate permutation when characters are repeated. We can avoid it by keeping track of the previous permutation. While printing, if the current permutation is same as previous permutation, we won't print it.

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Combinatorial Mathematical MathematicalAlgo permutation

Related Posts:

- Sum of all numbers that can be formed with permutations of n digits
- Longest common subsequence with permutations allowed
- Number of ways to make mobile lock pattern
- Count permutations that produce positive result
- Find all distinct subsets of a given set
- Heap's Algorithm for generating permutations
- All permutations of a string using iteration
- Permutations of a given string using STL

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 32 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

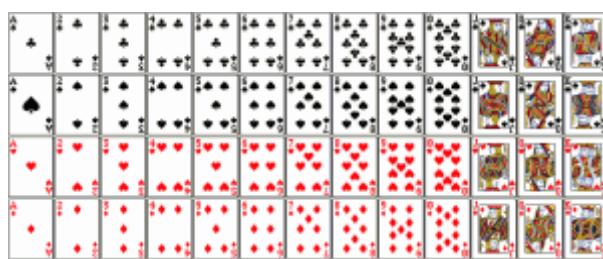
Google™ Custom Search



[Login/Register](#)

Shuffle a given array

Given an array, write a program to generate a random permutation of array elements. This question is also asked as “shuffle a deck of cards” or “randomize a given array”.



Let the given array be $arr[]$. A simple solution is to create an auxiliary array $temp[]$ which is initially a copy of $arr[]$. Randomly select an element from $temp[]$, copy the randomly selected element to $arr[0]$ and remove the selected element from $temp[]$. Repeat the same process n times and keep copying elements to $arr[1]$, $arr[2]$, The time complexity of this solution will be $O(n^2)$.

[Fisher–Yates shuffle Algorithm](#) works in $O(n)$ time complexity. The assumption here is, we are given a function `rand()` that generates random number in $O(1)$ time.

The idea is to start from the last element, swap it with a randomly selected element from the whole array (including last). Now consider the array from 0 to $n-2$ (size reduced by 1), and repeat the process till we hit the first element.

Following is the detailed algorithm

```
To shuffle an array a of n elements (indices 0..n-1):
for i from n - 1 downto 1 do
    j = random integer with 0 <= j <= i
    exchange a[j] and a[i]
```

Following is C++ implementation of this algorithm.

```
// C Program to shuffle a given array

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A utility function to swap two integers
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// A utility function to print an array
```

```

void printArray (int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// A function to generate a random permutation of arr[]
void randomize (int arr[], int n )
{
    // Use a different seed value so that we don't get same
    // result each time we run this program
    rand ( time(NULL) );

    // Start from the last element and swap one by one. We don't
    // need to run for the first element that's why i > 0
    for (int i = n-1; i > 0; i--)
    {
        // Pick a random index from 0 to i
        int j = rand() % (i+1);

        // Swap arr[i] with the element at random index
        swap(&arr[i], &arr[j]);
    }
}

// Driver program to test above function.
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int n = sizeof(arr)/ sizeof(arr[0]);
    randomize (arr, n);
    printArray(arr, n);

    return 0;
}

```

[Run on IDE](#)

Output:

7 8 4 6 3 1 2 5

The above function assumes that rand() generates a random number.

Time Complexity: O(n), assuming that the function rand() takes O(1) time.

How does this work?

The probability that ith element (including the last one) goes to last position is $1/n$, because we randomly pick an element in first iteration.

The probability that ith element goes to second last position can be proved to be $1/n$ by dividing it in two cases.

Case 1: $i = n-1$ (index of last element):

The probability of last element going to second last position is = (probability that last element doesn't stay at its original position) \times (probability that the index picked in previous step is picked again so that the last element is swapped)

So the probability = $((n-1)/n) \times (1/(n-1)) = 1/n$

Case 2: $0 < i < n-1$ (index of non-last):

The probability of ith element going to second position = (probability that ith element is not picked in previous iteration) \times (probability that ith element is picked in this iteration)

So the probability = $((n-1)/n) \times (1/(n-1)) = 1/n$

We can easily generalize above proof for any other position.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Randomized MathematicalAlgo

Related Posts:

- Program to generate CAPTCHA and verify user
- Find an index of maximum occurring element with equal probability
- Implement rand12() using rand6() in one line
- Implement rand3() using rand2()
- Generate 0 and 1 with 25% and 75% probability
- Randomized Algorithms | Set 0 (Mathematical Background)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)
- Primality Test | Set 2 (Fermat Method)

(Login to Rate and Mark)

3.2 Average Difficulty : **3.2/5.0**
Based on **37** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Space and time efficient Binomial Coefficient

Write a function that takes two parameters n and k and returns the value of Binomial Coefficient $C(n, k)$. For example, your function should return 6 for $n = 4$ and $k = 2$, and it should return 10 for $n = 5$ and $k = 2$.

We have discussed a $O(n*k)$ time and $O(k)$ extra space algorithm in [this](#) post. The value of $C(n, k)$ can be calculated in $O(k)$ time and $O(1)$ extra space.

$$\begin{aligned} C(n, k) &= n! / (n-k)! * k! \\ &= [n * (n-1) * \dots * 1] / [((n-k) * (n-k-1) * \dots * 1) * \\ &\quad (k * (k-1) * \dots * 1)] \end{aligned}$$

After simplifying, we get

$$C(n, k) = [n * (n-1) * \dots * (n-k+1)] / [k * (k-1) * \dots * 1]$$

Also, $C(n, k) = C(n, n-k)$ // we can change r to $n-r$ if $r > n-r$

Following implementation uses above formula to calculate $C(n, k)$

```
// Program to calculate C(n ,k)
#include <stdio.h>

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    int res = 1;

    // Since C(n, k) = C(n, n-k)
    if ( k > n - k )
        k = n - k;

    // Calculate value of [n * (n-1) *---* (n-k+1)] / [k * (k-1) *----* 1]
    for (int i = 0; i < k; ++i)
    {
        res *= (n - i);
        res /= (i + 1);
    }

    return res;
}

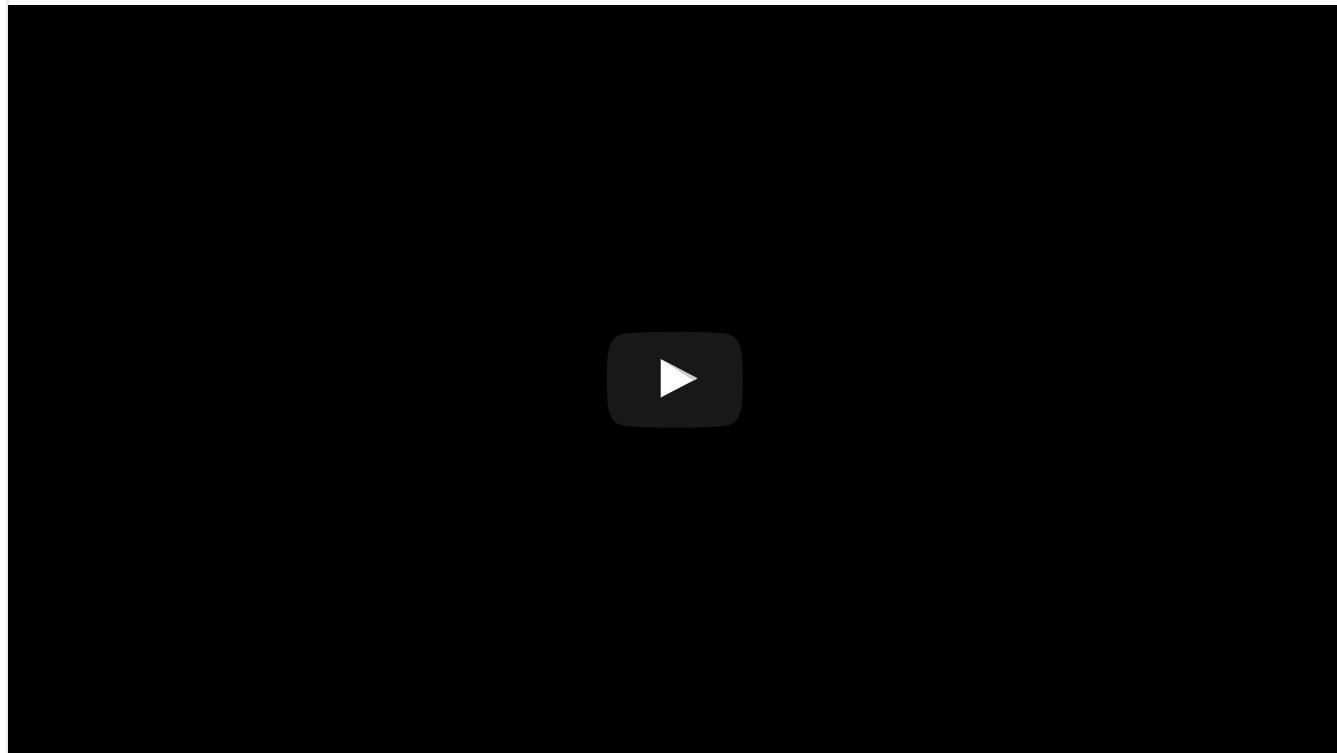
/* Drier program to test above function*/
int main()
{
    int n = 8, k = 2;
    printf ("Value of C(%d, %d) is %d ", n, k, binomialCoeff(n, k) );
    return 0;
}
```

[Run on IDE](#)

Value of $C(8, 2)$ is 28

Time Complexity: $O(k)$

Auxiliary Space: $O(1)$



This article is compiled by [Aashish Barnwal](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Mathematical

MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.2

Average Difficulty : 2.2/5.0
Based on 17 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Reservoir Sampling

Reservoir sampling is a family of randomized algorithms for randomly choosing k samples from a list of n items, where n is either a very large or unknown number. Typically n is large enough that the list doesn't fit into main memory. For example, a list of search queries in Google and Facebook.

So we are given a big array (or stream) of numbers (to simplify), and we need to write an efficient function to randomly select k numbers where $1 \leq k \leq n$. Let the input array be $stream[]$.

A **simple solution** is to create an array $reservoir[]$ of maximum size k . One by one randomly select an item from $stream[0..n-1]$. If the selected item is not previously selected, then put it in $reservoir[]$. To check if an item is previously selected or not, we need to search the item in $reservoir[]$. The time complexity of this algorithm will be $O(k^2)$. This can be costly if k is big. Also, this is not efficient if the input is in the form of a stream.

It can be solved in $O(n)$ time. The solution also suits well for input in the form of stream. The idea is similar to [this post](#). Following are the steps.

1) Create an array $reservoir[0..k-1]$ and copy first k items of $stream[]$ to it.

2) Now one by one consider all items from $(k+1)$ th item to n th item.

...a) Generate a random number from 0 to i where i is index of current item in $stream[]$. Let the generated random number is j .

...b) If j is in range 0 to $k-1$, replace $reservoir[j]$ with $arr[i]$

Following is C implementation of the above algorithm.

```
// An efficient program to randomly select k items from a stream of items
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A utility function to print an array
void printArray(int stream[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", stream[i]);
    printf("\n");
}

// A function to randomly select k items from stream[0..n-1].
void selectKItems(int stream[], int n, int k)
{
    int i; // index for elements in stream[]

    // reservoir[] is the output array. Initialize it with
    // first k elements from stream[]
    int reservoir[k];
    for (i = 0; i < k; i++)
        reservoir[i] = stream[i];
```

```

// Use a different seed value so that we don't get
// same result each time we run this program
rand(time(NULL));

// Iterate from the (k+1)th element to nth element
for ( ; i < n; i++)
{
    // Pick a random index from 0 to i.
    int j = rand() % (i+1);

    // If the randomly picked index is smaller than k, then replace
    // the element present at the index with new element from stream
    if (j < k)
        reservoir[j] = stream[i];
}

printf("Following are k randomly selected items \n");
printArray(reservoir, k);
}

// Driver program to test above function.
int main()
{
    int stream[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int n = sizeof(stream)/sizeof(stream[0]);
    int k = 5;
    selectKItems(stream, n, k);
    return 0;
}

```

[Run on IDE](#)

Output:

```

Following are k randomly selected items
6 2 11 8 12

```

Time Complexity: O(n)

How does this work?

To prove that this solution works perfectly, we must prove that the probability that any item $stream[i]$ where $0 \leq i < n$ will be in final $reservoir[]$ is k/n . Let us divide the proof in two cases as first k items are treated differently.

Case 1: For last $n-k$ stream items, i.e., for $stream[i]$ where $k \leq i < n$

For every such stream item $stream[i]$, we pick a random index from 0 to i and if the picked index is one of the first k indexes, we replace the element at picked index with $stream[i]$.

To simplify the proof, let us first consider the *last item*. The probability that the last item is in final reservoir = The probability that one of the first k indexes is picked for last item = k/n (the probability of picking one of the k items from a list of size n)

Let us now consider the *second last item*. The probability that the second last item is in final $reservoir[]$ = [Probability that one of the first k indexes is picked in iteration for $stream[n-2]$] X [Probability that the index picked in iteration for $stream[n-1]$ is not same as index picked for $stream[n-2]$] = $[k/(n-1)] * [(n-1)/n] = k/n$.

Similarly, we can consider other items for all stream items from $stream[n-1]$ to $stream[k]$ and generalize the proof.

Case 2: For first k stream items, i.e., for $stream[i]$ where $0 \leq i < k$

The first k items are initially copied to $reservoir[]$ and may be removed later in iterations for $stream[k]$ to $stream[n]$.

The probability that an item from $stream[0..k-1]$ is in final array = Probability that the item is not picked when items

$stream[k], stream[k+1], \dots, stream[n-1]$ are considered = $[k/(k+1)] \times [(k+1)/(k+2)] \times [(k+2)/(k+3)] \times \dots \times [(n-1)/n] = k/n$

References:

http://en.wikipedia.org/wiki/Reservoir_sampling

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

[Articles](#) [Randomized](#) [MathematicalAlgo](#)

Related Posts:

- [SQL | DROP, TRUNCATE](#)
- [SQL | CREATE](#)
- [SQL | Join \(Cartesian Join & Self Join\)](#)
- [SQL | Union Clause](#)
- [SQL | Join \(Inner, Left, Right and Full Joins\)](#)
- [SQL | Wildard operators](#)
- [SQL | Aliases](#)
- [SQL | ORDER BY](#)

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 36 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Pascal's Triangle

Pascal's triangle is a triangular array of the binomial coefficients. Write a function that takes an integer value n as input and prints first n lines of the Pascal's triangle. Following are the first 6 rows of Pascal's Triangle.

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

We strongly recommend that you click here and practice it, before moving on to the solution.

Method 1 (O(n^3) time complexity)

Number of entries in every line is equal to line number. For example, the first line has "1", the second line has "1 1", the third line has "1 2 1",.. and so on. Every entry in a line is value of a [Binomial Coefficient](#). The value of i th entry in line number $line$ is $C(line, i)$. The value can be calculated using following formula.

```
C(line, i) = line! / ( (line-i)! * i! )
```

A simple method is to run two loops and calculate the value of Binomial Coefficient in inner loop.

```

// A simple O(n^3) program for Pascal's Triangle
#include <stdio.h>

// See http://www.geeksforgeeks.org/archives/25621 for details of this function
int binomialCoeff(int n, int k);

// Function to print first n lines of Pascal's Triangle
void printPascal(int n)
{
    // Iterate through every line and print entries in it
    for (int line = 0; line < n; line++)
    {
        // Every line has number of integers equal to line number
        for (int i = 0; i <= line; i++)
            printf("%d ", binomialCoeff(line, i));
        printf("\n");
    }
}

```

```
// See http://www.geeksforgeeks.org/archives/25621 for details of this function
int binomialCoeff(int n, int k)
{
    int res = 1;
    if (k > n - k)
        k = n - k;
    for (int i = 0; i < k; ++i)
    {
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}

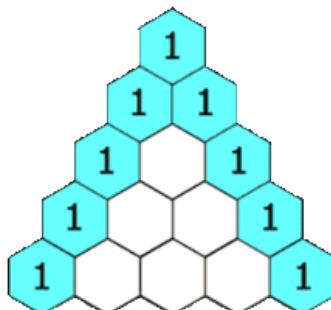
// Driver program to test above function
int main()
{
    int n = 7;
    printPascal(n);
    return 0;
}
```

[Run on IDE](#)

Time complexity of this method is $O(n^3)$. Following are optimized methods.

Method 2($O(n^2)$ time and $O(n^2)$ extra space)

If we take a closer at the triangle, we observe that every entry is sum of the two values above it. So we can create a 2D array that stores previously generated values. To generate a value in a line, we can use the previously stored values from array.



```
// A O(n^2) time and O(n^2) extra space method for Pascal's Triangle
void printPascal(int n)
{
    int arr[n][n]; // An auxiliary array to store generated pascal triangle values

    // Iterate through every line and print integer(s) in it
    for (int line = 0; line < n; line++)
    {
        // Every line has number of integers equal to line number
        for (int i = 0; i <= line; i++)
        {
            // First and last values in every row are 1
            if (line == i || i == 0)
                arr[line][i] = 1;
            else // Other values are sum of values just above and left of above
                arr[line][i] = arr[line-1][i-1] + arr[line-1][i];
            printf("%d ", arr[line][i]);
        }
        printf("\n");
    }
}
```

[Run on IDE](#)

This method can be optimized to use $O(n)$ extra space as we need values only from previous row. So we can create an auxiliary array of size n and overwrite values. Following is another method uses only $O(1)$ extra space.

Method 3 ($O(n^2)$ time and $O(1)$ extra space)

This method is based on method 1. We know that i th entry in a line number $line$ is Binomial Coefficient $C(line, i)$ and all lines start with value 1. The idea is to calculate $C(line, i)$ using $C(line, i-1)$. It can be calculated in $O(1)$ time using the following.

```
C(line, i) = line! / ( (line-i)! * i! )
C(line, i-1) = line! / ( (line - i + 1)! * (i-1)! )
We can derive following expression from above two expressions.
C(line, i) = C(line, i-1) * (line - i + 1) / i
```

So $C(line, i)$ can be calculated from $C(line, i-1)$ in $O(1)$ time

```
// A O(n^2) time and O(1) extra space function for Pascal's Triangle
void printPascal(int n)
{
    for (int line = 1; line <= n; line++)
    {
        int C = 1; // used to represent C(line, i)
        for (int i = 1; i <= line; i++)
        {
            printf("%d ", C); // The first value in a line is always 1
            C = C * (line - i) / i;
        }
        printf("\n");
    }
}
```

[Run on IDE](#)

So method 3 is the best method among all, but it may cause integer overflow for large values of n as it multiplies two integers to obtain values.

This article is compiled by **Rahul** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Mathematical MathematicalAlgo pattern-printing

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 20 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Select a random number from stream, with O(1) space

Given a stream of numbers, generate a random number from the stream. You are allowed to use only O(1) space and the input is in the form of stream, so can't store the previously seen numbers.

So how do we generate a random number from the whole stream such that the probability of picking any number is 1/n. with O(1) extra space? This problem is a variation of [Reservoir Sampling](#). Here the value of k is 1.

1) Initialize 'count' as 0, 'count' is used to store count of numbers seen so far in stream.

2) For each number 'x' from stream, do following

.....a) Increment 'count' by 1.

.....b) If count is 1, set result as x, and return result.

.....c) Generate a random number from 0 to 'count-1'. Let the generated random number be i.

.....d) If i is equal to 'count – 1', update the result as x.

```
// An efficient program to randomly select a number from stream of numbers.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A function to randomly select a item from stream[0], stream[1], .. stream[i-1]
int selectRandom(int x)
{
    static int res;      // The resultant random number
    static int count = 0; //Count of numbers visited so far in stream

    count++; // increment count of numbers seen so far

    // If this is the first element from stream, return it
    if (count == 1)
        res = x;
    else
    {
        // Generate a random number from 0 to count - 1
        int i = rand() % count;

        // Replace the prev random number with new number with 1/count probability
        if (i == count - 1)
            res = x;
    }
    return res;
}

// Driver program to test above function.
int main()
{
    int stream[] = {1, 2, 3, 4};
    int n = sizeof(stream)/sizeof(stream[0]);

    // Use a different seed value for every run.
}
```

```

strand(time(NULL));
for (int i = 0; i < n; ++i)
    printf("Random number from first %d numbers is %d \n",
           i+1, selectRandom(stream[i]));
return 0;
}

```

[Run on IDE](#)

Output:

```

Random number from first 1 numbers is 1
Random number from first 2 numbers is 1
Random number from first 3 numbers is 3
Random number from first 4 numbers is 4

```

Auxiliary Space: O(1)

How does this work

We need to prove that every element is picked with $1/n$ probability where n is the number of items seen so far. For every new stream item x , we pick a random number from 0 to 'count -1', if the picked number is 'count-1', we replace the previous result with x .

To simplify proof, let us first consider the last element, the last element replaces the previously stored result with $1/n$ probability. So probability of getting last element as result is $1/n$.

Let us now talk about second last element. When second last element processed first time, the probability that it replaced the previous result is $1/(n-1)$. The probability that previous result stays when nth item is considered is $(n-1)/n$. So probability that the second last element is picked in last iteration is $[1/(n-1)] * [(n-1)/n]$ which is $1/n$.

Similarly, we can prove for third last element and others.

References:

[Reservoir Sampling](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Randomized](#)[MathematicalAlgo](#)[Random Algorithms](#)

Related Posts:

- Program to generate CAPTCHA and verify user
- Find an index of maximum occurring element with equal probability
- Implement rand12() using rand6() in one line
- Implement rand3() using rand2()
- Generate 0 and 1 with 25% and 75% probability
- Randomized Algorithms | Set 0 (Mathematical Background)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)
- Primality Test | Set 2 (Fermat Method)

(Login to Rate and Mark)



Average Difficulty : **4/5.0**
Based on **10** vote(s)



Add to TODO List

 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find the largest multiple of 2, 3 and 5

An array of size n is given. The array contains digits from 0 to 9. Generate the largest number using the digits in the array such that the number is divisible by 2, 3 and 5.

For example, if the arrays is {1, 8, 7, 6, 0}, output must be: 8760. And if the arrays is {7, 7, 7, 6}, output must be: "no number can be formed".

Source: [Amazon Interview | Set 7](#)

This problem is a variation of "Find the largest multiple of 3".

Since the number has to be divisible by 2 and 5, it has to have last digit as 0. So if the given array doesn't contain any zero, then no solution exists.

Once a 0 is available, extract 0 from the given array. Only thing left is, the number should be is divisible by 3 and the largest of all. Which has been discussed [here](#).

Thanks to [shashank](#) for suggesting this solution. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

ⓘ ✕

Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

[Start Now](#)


|

HARVARD
UNIVERSITY

[GATE CS Corner](#) [Company Wise Coding Practice](#)

[Mathematical](#)[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.1

Average Difficulty : **2.1/5.0**
Based on **6** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Efficient program to calculate e^x

The value of [Exponential Function](#) e^x can be expressed using following [Taylor Series](#).

$$e^x = 1 + x/1! + x^2/2! + x^3/3! + \dots$$

How to efficiently calculate the sum of above series?

The series can be re-written as

$$e^x = 1 + (x/1) (1 + (x/2) (1 + (x/3) (\dots)))$$

Let the sum needs to be calculated for n terms, we can calculate sum using following loop.

```
for (i = n - 1, sum = 1; i > 0; --i )
    sum = 1 + x * sum / i;
```

Following is implementation of the above idea.

```
// Efficient program to calculate e raise to the power x
#include <stdio.h>

// Returns approximate value of e^x using sum of first n terms of Taylor Series
float exponential(int n, float x)
{
    float sum = 1.0f; // initialize sum of series

    for (int i = n - 1; i > 0; --i )
        sum = 1 + x * sum / i;

    return sum;
}

// Driver program to test above function
int main()
{
    int n = 10;
    float x = 1.0f;
    printf("e^x = %f", exponential(n, x));
    return 0;
}
```

[Run on IDE](#)

Output:

e^x = 2.718282

This article is compiled by **Rahul** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Introduction to Computer Science

ENROLLMENT IS OPEN TO EVERYONE

Start Now

edX | HARVARD UNIVERSITY

GATE CS Corner Company Wise Coding Practice

Mathematical MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.6 Average Difficulty : **2.6/5.0**
Based on **5** vote(s)

Add to TODO List
 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Measure one litre using two vessels and infinite water supply

There are two vessels of capacities 'a' and 'b' respectively. We have infinite water supply. Give an efficient algorithm to make exactly 1 litre of water in one of the vessels. You can throw all the water from any vessel any point of time. Assume that 'a' and 'b' are [Coprimes](#).

Following are the steps:

Let V1 be the vessel of capacity 'a' and V2 be the vessel of capacity 'b' and 'a' is smaller than 'b'.

1) Do following while the amount of water in V1 is not 1.

....**a)** If V1 is empty, then completely fill V1

....**b)** Transfer water from V1 to V2. If V2 becomes full, then keep the remaining water in V1 and empty V2

2) V1 will have 1 litre after termination of loop in step 1. Return.

Following is C++ implementation of the above algorithm.

```
/* Sample run of the Algo for V1 with capacity 3 and V2 with capacity 7
1. Fill V1: V1 = 3, V2 = 0
2. Transfer from V1 to V2, and fill V1: V1 = 3, V2 = 3
3. Transfer from V1 to V2, and fill V1: V1 = 3, V2 = 6
4. Transfer from V1 to V2, and empty V2: V1 = 2, V2 = 0
5. Transfer from V1 to V2, and fill V1: V1 = 3, V2 = 2
6. Transfer from V1 to V2, and fill V1: V1 = 3, V2 = 5
7. Stop as V1 now contains 1 litre.
```

Note that V2 was made empty in steps 3 and 6 because it became full */

```
#include <iostream>
using namespace std;

// A utility function to get GCD of two numbers
int gcd(int a, int b) { return b? gcd(b, a % b) : a; }

// Class to represent a Vessel
class Vessel
{
    // A vessel has capacity, and current amount of water in it
    int capacity, current;
public:
    // Constructor: initializes capacity as given, and current as 0
    Vessel(int capacity) { this->capacity = capacity; current = 0; }

    // The main function to fill one litre in this vessel. Capacity of V2
    // must be greater than this vessel and two capacities must be co-prime
    void makeOneLitre(Vessel &V2);

    // Fills vessel with given amount and returns the amount of water
    // transferred to it. If the vessel becomes full, then the vessel
    // is made empty.
    int transfer(int amount);
```

```

};

// The main function to fill one litre in this vessel. Capacity
// of V2 must be greater than this vessel and two capacities
// must be coprime
void Vessel:: makeOneLitre(Vessel &V2)
{
    // solution exists iff a and b are co-prime
    if (gcd(capacity, V2.capacity) != 1)
        return;

    while (current != 1)
    {
        // fill A (smaller vessel)
        if (current == 0)
            current = capacity;

        cout << "Vessel 1: " << current << "    Vessel 2: "
            << V2.current << endl;

        // Transfer water from V1 to V2 and reduce current of V1 by
        // the amount equal to transferred water
        current = current - V2.transfer(current);
    }

    // Finally, there will be 1 litre in vessel 1
    cout << "Vessel 1: " << current << "    Vessel 2: "
        << V2.current << endl;
}

// Fills vessel with given amount and returns the amount of water
// transferred to it. If the vessel becomes full, then the vessel
// is made empty
int Vessel::transfer(int amount)
{
    // If the vessel can accommodate the given amount
    if (current + amount < capacity)
    {
        current += amount;
        return amount;
    }

    // If the vessel cannot accommodate the given amount, then
    // store the amount of water transferred
    int transferred = capacity - current;

    // Since the vessel becomes full, make the vessel
    // empty so that it can be filled again
    current = 0;

    return transferred;
}

// Driver program to test above function
int main()
{
    int a = 3, b = 7; // a must be smaller than b

    // Create two vessels of capacities a and b
    Vessel V1(a), V2(b);

    // Get 1 litre in first vessel
    V1.makeOneLitre(V2);

    return 0;
}

```

Run on IDE

Output:

```

Vessel 1: 3   Vessel 2: 0
Vessel 1: 3   Vessel 2: 3
Vessel 1: 3   Vessel 2: 6
Vessel 1: 2   Vessel 2: 0
Vessel 1: 3   Vessel 2: 2
Vessel 1: 3   Vessel 2: 5
Vessel 1: 1   Vessel 2: 0

```

How does this work?

To prove that the algorithm works, we need to proof that after certain number of iterations in the while loop, we will get 1 litre in V1.

Let 'a' be the capacity of vessel V1 and 'b' be the capacity of V2. Since we repeatedly transfer water from V1 to V2 until V2 becomes full, we will have ' $a - b \pmod{a}$ ' water in V1 when V2 becomes full first time . Once V2 becomes full, it is emptied. We will have ' $a - 2b \pmod{a}$ ' water in V1 when V2 is full second time. We repeat the above steps, and get ' $a - nb \pmod{a}$ ' water in V1 after the vessel V2 is filled and emptied 'n' times. We need to prove that the value of ' $a - nb \pmod{a}$ ' will be 1 for a finite integer 'n'. To prove this, let us consider the following property of coprime numbers.

For any two **coprime integers** 'a' and 'b', the integer 'b' has a **multiplicative inverse** modulo 'a'. In other words, there exists an integer 'y' such that ' $b^*y \equiv 1 \pmod{a}$ ' (See 3rd point [here](#)). After ' $(a - 1)^*y$ ' iterations, we will have ' $a - [(a - 1)^*y * b] \pmod{a}$ ' water in V1, the value of this expression is ' $a - [(a - 1) * 1] \pmod{a}$ ' which is 1. So the algorithm converges and we get 1 litre in V1.

This article is compiled by [Aashish Barnwal](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#) [modular-arithmetic](#)

Related Posts:

- [Nth character in Concatenated Decimal String](#)
- [Find the highest occurring digit in prime numbers in a range](#)

- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4.5

Average Difficulty : **4.5/5.0**
Based on **15** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Efficient program to print all prime factors of a given number

Given a number n, write an efficient function to print all prime factors of n. For example, if the input number is 12, then output should be “2 2 3”. And if the input number is 315, then output should be “3 3 5 7”.

We strongly recommend that you click here and practice it, before moving on to the solution.

Following are the steps to find all prime factors.

- 1) While n is divisible by 2, print 2 and divide n by 2.
- 2) After step 1, n must be odd. Now start a loop from i = 3 to square root of n. While i divides n, print i and divide n by i, increment i by 2 and continue.
- 3) If n is a prime number and is greater than 2, then n will not become 1 by above two steps. So print n if it is greater than 2.

```
// Program to print all prime factors
# include <stdio.h>
# include <math.h>

// A function to print all prime factors of a given number n
void primeFactors(int n)
{
    // Print the number of 2s that divide n
    while (n%2 == 0)
    {
        printf("%d ", 2);
        n = n/2;
    }

    // n must be odd at this point. So we can skip one element (Note i = i +2)
    for (int i = 3; i <= sqrt(n); i = i+2)
    {
        // While i divides n, print i and divide n
        while (n%i == 0)
        {
            printf("%d ", i);
            n = n/i;
        }
    }

    // This condition is to handle the case when n is a prime number
    // greater than 2
    if (n > 2)
        printf("%d ", n);
}
```

```

        printf ("%d ", n);
    }

/* Driver program to test above function */
int main()
{
    int n = 315;
    primeFactors(n);
    return 0;
}

```

[Run on IDE](#)

Java

```

// Program to print all prime factors
import java.io.*;
import java.lang.Math;

class GFG
{
    // A function to print all prime factors
    // of a given number n
    public static void primeFactors(int n)
    {
        // Print the number of 2s that divide n
        while (n%2==0)
        {
            System.out.print(2 + " ");
            n /= 2;
        }

        // n must be odd at this point. So we can
        // skip one element (Note i = i +2)
        for (int i = 3; i <= Math.sqrt(n); i+= 2)
        {
            // While i divides n, print i and divide n
            while (n%i == 0)
            {
                System.out.print(i + " ");
                n /= i;
            }
        }

        // This condition is to handle the case when
        // n is a prime number greater than 2
        if (n > 2)
            System.out.print(n);
    }

    public static void main (String[] args)
    {
        int n = 315;
        primeFactors(n);
    }
}

```

[Run on IDE](#)

Output:

3 3 5 7

How does this work?

The steps 1 and 2 take care of composite numbers and step 3 takes care of prime numbers. To prove that the

complete algorithm works, we need to prove that steps 1 and 2 actually take care of composite numbers. This is clear that step 1 takes care of even numbers. And after step 1, all remaining prime factor must be odd (difference of two prime factors must be at least 2), this explains why i is incremented by 2.

Now the main part is, the loop runs till square root of n not till. To prove that this optimization works, let us consider the following property of composite numbers.

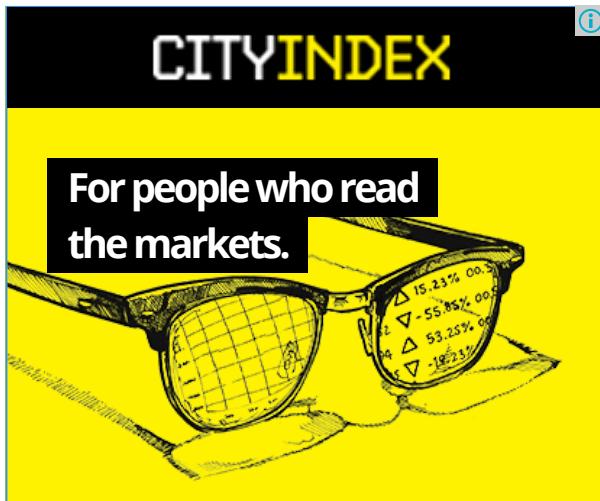
Every composite number has at least one prime factor less than or equal to square root of itself.

This property can be proved using counter statement. Let a and b be two factors of n such that $a * b = n$. If both are greater than \sqrt{n} , then $a * b > \sqrt{n} * \sqrt{n}$, which contradicts the expression " $a * b = n$ ".

In step 2 of the above algorithm, we run a loop and do following in loop

- Find the least prime factor i (must be less than \sqrt{n})
- Remove all occurrences i from n by repeatedly dividing n by i.
- Repeat steps a and b for divided n and $i = i + 2$. The steps a and b are repeated till n becomes either 1 or a prime number.

Thanks to **Vishwas Garg** for suggesting the above algorithm. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

Mathematical combinatorics MathematicalAlgo prime-factor

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

[\(Login to Rate and Mark\)](#)**2.3**Average Difficulty : **2.3/5.0**
Based on **42** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Print all possible combinations of r elements in a given array of size n

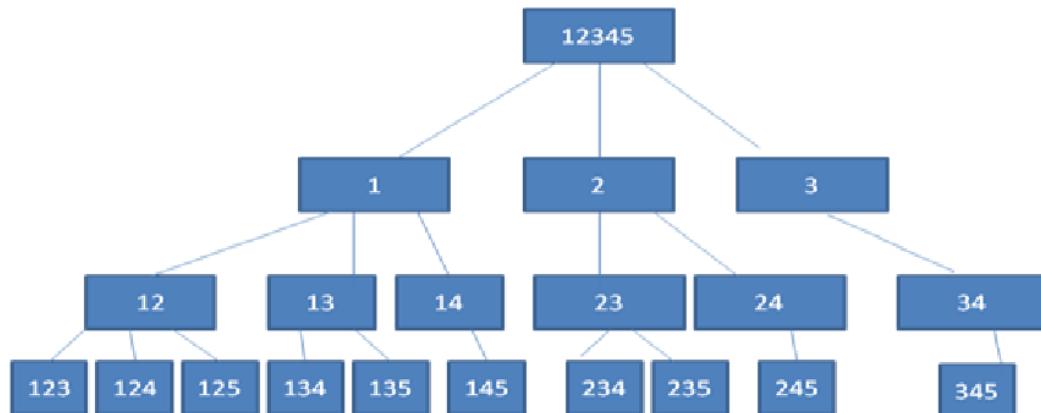
Given an array of size n, generate and print all possible combinations of r elements in array. For example, if input array is {1, 2, 3, 4} and r is 2, then output should be {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4} and {3, 4}.

Following are two methods to do this.

Method 1 (Fix Elements and Recur)

We create a temporary array 'data[]' which stores all outputs one by one. The idea is to start from first index (index = 0) in data[], one by one fix elements at this index and recur for remaining indexes. Let the input array be {1, 2, 3, 4, 5} and r be 3. We first fix 1 at index 0 in data[], then recur for remaining indexes, then we fix 2 at index 0 and recur. Finally, we fix 3 and recur for remaining indexes. When number of elements in data[] becomes equal to r (size of a combination), we print data[].

Following diagram shows recursion tree for same input.



Following is C++ implementation of above approach.

```

// Program to print all combination of size r in an array of size n
#include <stdio.h>
void combinationUtil(int arr[], int data[], int start, int end,
                     int index, int r);

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination one by one
    int data[r];
  
```

```

// Print all combination using temporary array 'data[]'
combinationUtil(arr, data, 0, n-1, 0, r);
}

/* arr[] ----> Input Array
   data[] ----> Temporary array to store current combination
   start & end ----> Starting and Ending indexes in arr[]
   index ----> Current index in data[]
   r ----> Size of a combination to be printed */
void combinationUtil(int arr[], int data[], int start, int end,
                     int index, int r)
{
    // Current combination is ready to be printed, print it
    if (index == r)
    {
        for (int j=0; j<r; j++)
            printf("%d ", data[j]);
        printf("\n");
        return;
    }

    // replace index with all possible elements. The condition
    // "end-i+1 >= r-index" makes sure that including one element
    // at index will make a combination with remaining elements
    // at remaining positions
    for (int i=start; i<=end && end-i+1 >= r-index; i++)
    {
        data[index] = arr[i];
        combinationUtil(arr, data, i+1, end, index+1, r);
    }
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCombination(arr, n, r);
}

```

[Run on IDE](#)

Java

```

// Java program to print all combination of size r in an array of size n
import java.io.*;

class Permutation {

    /* arr[] ----> Input Array
       data[] ----> Temporary array to store current combination
       start & end ----> Starting and Ending indexes in arr[]
       index ----> Current index in data[]
       r ----> Size of a combination to be printed */
    static void combinationUtil(int arr[], int data[], int start,
                                int end, int index, int r)
    {
        // Current combination is ready to be printed, print it
        if (index == r)
        {
            for (int j=0; j<r; j++)
                System.out.print(data[j]+" ");
            System.out.println("");
            return;
        }

        // replace index with all possible elements. The condition
        // "end-i+1 >= r-index" makes sure that including one element
        // at index will make a combination with remaining elements
    }
}

```

```

// at remaining positions
for (int i=start; i<=end && end-i+1 >= r-index; i++)
{
    data[index] = arr[i];
    combinationUtil(arr, data, i+1, end, index+1, r);
}

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
static void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination one by one
    int data[] = new int[r];

    // Print all combination using temporary array 'data[]'
    combinationUtil(arr, data, 0, n-1, 0, r);
}

/*Driver function to check for above function*/
public static void main (String[] args) {
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = arr.length;
    printCombination(arr, n, r);
}
}

/* This code is contributed by Devesh Agrawal */

```

[Run on IDE](#)

Output:

```

1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5

```

How to handle duplicates?

Note that the above method doesn't handle duplicates. For example, if input array is {1, 2, 1} and r is 2, then the program prints {1, 2} and {2, 1} as two different combinations. We can avoid duplicates by adding following two additional things to above code.

- 1) Add code to sort the array before calling combinationUtil() in printCombination()
- 2) Add following lines at the end of for loop in combinationUtil()

```

// Since the elements are sorted, all occurrences of an element
// must be together
while (arr[i] == arr[i+1])
    i++;

```

See [this](#) for an implementation that handles duplicates.

Method 2 (Include and Exclude every element)

Like the above method, We create a temporary array `data[]`. The idea here is similar to [Subset Sum Problem](#). We one by one consider every element of input array, and recur for two cases:

1) The element is included in current combination (We put the element in `data[]` and increment next available index in `data[]`)

2) The element is excluded in current combination (We do not put the element and do not change index)

When number of elements in `data[]` become equal to r (size of a combination), we print it.

This method is mainly based on [Pascal's Identity](#), i.e. $n_c_r = n-1_c_r + n-1_c_{r-1}$

Following is C++ implementation of method 2.

```
// Program to print all combination of size r in an array of size n
#include<stdio.h>
void combinationUtil(int arr[],int n,int r,int index,int data[],int i);

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination one by one
    int data[r];

    // Print all combination using temporary array 'data[]'
    combinationUtil(arr, n, r, 0, data, 0);
}

/* arr[] --> Input Array
   n      --> Size of input array
   r      --> Size of a combination to be printed
   index  --> Current index in data[]
   data[] --> Temporary array to store current combination
   i      --> index of current element in arr[]      */
void combinationUtil(int arr[], int n, int r, int index, int data[], int i)
{
    // Current combination is ready, print it
    if (index == r)
    {
        for (int j=0; j<r; j++)
            printf("%d ",data[j]);
        printf("\n");
        return;
    }

    // When no more elements are there to put in data[]
    if (i >= n)
        return;

    // current is included, put next at next location
    data[index] = arr[i];
    combinationUtil(arr, n, r, index+1, data, i+1);

    // current is excluded, replace it with next (Note that
    // i+1 is passed, but index is not changed)
    combinationUtil(arr, n, r, index, data, i+1);
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
```

```

int r = 3;
int n = sizeof(arr)/sizeof(arr[0]);
printCombination(arr, n, r);
return 0;
}

```

[Run on IDE](#)

Java

```

// Java program to print all combination of size r in an array of size n
import java.io.*;

class Permutation {

    /* arr[] ---> Input Array
    data[] ---> Temporary array to store current combination
    start & end ---> Starting and Ending indexes in arr[]
    index ---> Current index in data[]
    r ---> Size of a combination to be printed */
    static void combinationUtil(int arr[], int n, int r, int index,
                                int data[], int i)
    {
        // Current combination is ready to be printed, print it
        if (index == r)
        {
            for (int j=0; j<r; j++)
                System.out.print(data[j]+" ");
            System.out.println("");
            return;
        }

        // When no more elements are there to put in data[]
        if (i >= n)
            return;

        // current is included, put next at next location
        data[index] = arr[i];
        combinationUtil(arr, n, r, index+1, data, i+1);

        // current is excluded, replace it with next (Note that
        // i+1 is passed, but index is not changed)
        combinationUtil(arr, n, r, index, data, i+1);
    }

    // The main function that prints all combinations of size r
    // in arr[] of size n. This function mainly uses combinationUtil()
    static void printCombination(int arr[], int n, int r)
    {
        // A temporary array to store all combination one by one
        int data[] = new int[r];

        // Print all combination using temporary array 'data[]'
        combinationUtil(arr, n, r, 0, data, 0);
    }

    /*Driver function to check for above function*/
    public static void main (String[] args) {
        int arr[] = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.length;
        printCombination(arr, n, r);
    }
}

/* This code is contributed by Devesh Agrawal */

```

[Run on IDE](#)

Output:

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

How to handle duplicates in method 2?

Like method 1, we can follow two things to handle duplicates.

- 1) Add code to sort the array before calling combinationUtil() in printCombination()
- 2) Add following lines between two recursive calls of combinationUtil() in combinationUtil()

```
// Since the elements are sorted, all occurrences of an element
// must be together
while (arr[i] == arr[i+1])
    i++;
```

See [this](#) for an implementation that handles duplicates.

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



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#) [Recursion](#)

Related Posts:

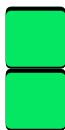
- Nth character in Concatenated Decimal String

- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 60 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Random number generator in arbitrary probability distribution fashion

Given n numbers, each with some frequency of occurrence. Return a random number with probability proportional to its frequency of occurrence.

Example:

Let following be the given numbers.

```
arr[] = {10, 30, 20, 40}
```

Let following be the frequencies of given numbers.

```
freq[] = {1, 6, 2, 1}
```

The output should be

```
10 with probability 1/10
30 with probability 6/10
20 with probability 2/10
40 with probability 1/10
```

It is quite clear that the simple random number generator won't work here as it doesn't keep track of the frequency of occurrence.

We need to somehow transform the problem into a problem whose solution is known to us.

One simple method is to take an auxiliary array (say aux[]) and duplicate the numbers according to their frequency of occurrence. Generate a random number(say r) between 0 to Sum-1(including both), where Sum represents summation of frequency array (freq[] in above example). Return the random number aux[r] (Implementation of this method is left as an exercise to the readers).

The limitation of the above method discussed above is huge memory consumption when frequency of occurrence is high. If the input is 997, 8761 and 1, this method is clearly not efficient.

How can we reduce the memory consumption? Following is detailed algorithm that uses O(n) extra space where n is number of elements in input arrays.

1. Take an auxiliary array (say prefix[]) of size n.
2. Populate it with prefix sum, such that prefix[i] represents sum of numbers from 0 to i.
3. Generate a random number(say r) between 1 to Sum(including both), where Sum represents summation of input frequency array.
4. Find index of Ceil of random number generated in step #3 in the prefix array. Let the index be indexc.
5. Return the random number arr[indexc], where arr[] contains the input n numbers.

Before we go to the implementation part, let us have quick look at the algorithm with an example:

`arr[]: {10, 20, 30}`

`freq[]: {2, 3, 1}`

`Prefix[]: {2, 5, 6}`

Since last entry in prefix is 6, all possible values of r are [1, 2, 3, 4, 5, 6]

1: Ceil is 2. Random number generated is 10.

2: Ceil is 2. Random number generated is 10.

3: Ceil is 5. Random number generated is 20.

4: Ceil is 5. Random number generated is 20.

5: Ceil is 5. Random number generated is 20.

6: Ceil is 6. Random number generated is 30.

In the above example

10 is generated with probability 2/6.

20 is generated with probability 3/6.

30 is generated with probability 1/6.

How does this work?

Any number `input[i]` is generated as many times as its frequency of occurrence because there exists count of integers in range(`prefix[i - 1], prefix[i]`) is `input[i]`. Like in the above example 3 is generated thrice, as there exists 3 integers 3, 4 and 5 whose ceil is 5.

```
//C program to generate random numbers according to given frequency distribution
#include <stdio.h>
#include <stdlib.h>

// Utility function to find ceiling of r in arr[1..h]
int findCeil(int arr[], int r, int l, int h)
{
    int mid;
    while (l < h)
    {
        mid = l + ((h - 1) >> 1); // Same as mid = (l+h)/2
        (r > arr[mid]) ? (l = mid + 1) : (h = mid);
    }
    return (arr[l] >= r) ? l : -1;
}

// The main function that returns a random number from arr[] according to
// distribution array defined by freq[]. n is size of arrays.
int myRand(int arr[], int freq[], int n)
{
    // Create and fill prefix array
    int prefix[n], i;
    prefix[0] = freq[0];
    for (i = 1; i < n; ++i)
        prefix[i] = prefix[i - 1] + freq[i];

    // prefix[n-1] is sum of all frequencies. Generate a random number
    // with value from 1 to this sum
    int r = (rand() % prefix[n - 1]) + 1;

    // Find index of ceiling of r in prefix array
    int indexc = findCeil(prefix, r, 0, n - 1);
    return arr[indexc];
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4};
    int freq[] = {10, 5, 20, 100};
    int i, n = sizeof(arr) / sizeof(arr[0]);

    // Use a different seed value for every run.
}
```

```

strand(time(NULL));

// Let us generate 10 random numbers according to
// given distribution
for (i = 0; i < 5; i++)
    printf("%d\n", myRand(arr, freq, n));

return 0;
}

```

[Run on IDE](#)

Output: May be different for different runs

```

4
3
4
4
4

```

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

Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Randomized](#) [MathematicalAlgo](#)

Related Posts:

- Program to generate CAPTCHA and verify user
- Find an index of maximum occurring element with equal probability
- Implement rand12() using rand6() in one line
- Implement rand3() using rand2()
- Generate 0 and 1 with 25% and 75% probability
- Randomized Algorithms | Set 0 (Mathematical Background)
- Randomized Algorithms | Set 3 (1/2 Approximate Median)
- Primality Test | Set 2 (Fermat Method)

(Login to Rate and Mark)

3.8

Average Difficulty : **3.8/5.0**
Based on **10** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

How to check if a given number is Fibonacci number?

Given a number 'n', how to check if n is a Fibonacci number.

We strongly recommend that you click here and practice it, before moving on to the solution.

A simple way is to [generate Fibonacci numbers](#) until the generated number is greater than or equal to 'n'. Following is an interesting property about Fibonacci numbers that can also be used to check if a given number is Fibonacci or not.

A number is Fibonacci if and only if one or both of $(5n^2 + 4)$ or $(5n^2 - 4)$ is a perfect square (Source: [Wiki](#)). Following is a simple program based on this concept.

```
// C++ program to check if x is a perfect square
#include <iostream>
#include <math.h>
using namespace std;

// A utility function that returns true if x is perfect square
bool isPerfectSquare(int x)
{
    int s = sqrt(x);
    return (s*s == x);
}

// Returns true if n is a Fibonacci Number, else false
bool isFibonacci(int n)
{
    // n is Fibonacci if one of 5*n*n + 4 or 5*n*n - 4 or both
    // is a perfect square
    return isPerfectSquare(5*n*n + 4) ||
           isPerfectSquare(5*n*n - 4);
}

// A utility function to test above functions
int main()
{
    for (int i = 1; i <= 10; i++)
        isFibonacci(i)? cout << i << " is a Fibonacci Number \n":
                           cout << i << " is not a Fibonacci Number \n" ;
    return 0;
}
```

[Run on IDE](#)

Python

```
# python program to check if x is a perfect square
import math

# A utility function that returns true if x is perfect square
def isPerfectSquare(x):
    s = int(math.sqrt(x))
    return s*s == x

# Returns true if n is a Fibinacci Number, else false
def isFibonacci(n):

    # n is Fibinacci if one of 5*n*n + 4 or 5*n*n - 4 or both
    # is a perfect square
    return isPerfectSquare(5*n*n + 4) or isPerfectSquare(5*n*n - 4)

# A utility function to test above functions
for i in range(1,11):
    if (isFibonacci(i) == True):
        print i,"is a Fibonacci Number"
    else:
        print i,"is a not Fibonacci Number "
```

[Run on IDE](#)

Output:

```
1 is a Fibonacci Number
2 is a Fibonacci Number
3 is a Fibonacci Number
4 is a not Fibonacci Number
5 is a Fibonacci Number
6 is a not Fibonacci Number
7 is a not Fibonacci Number
8 is a Fibonacci Number
9 is a not Fibonacci Number
10 is a not Fibonacci Number
```

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

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Mathematical | Fibonacci | MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 16 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Russian Peasant Multiplication

Given two integers, write a function to multiply them without using multiplication operator.

There are many other ways to multiply two numbers (For example, see [this](#)). One interesting method is the **Russian peasant algorithm**. The idea is to double the first number and halve the second number repeatedly till the second number doesn't become 1. In the process, whenever the second number becomes odd, we add the first number to result (result is initialized as 0)

The following is simple algorithm.

```
Let the two given numbers be 'a' and 'b'
1) Initialize result 'res' as 0.
2) Do following while 'b' is greater than 0
   a) If 'b' is odd, add 'a' to 'res'
   b) Double 'a' and halve 'b'
3) Return 'res'.
```

```
#include <iostream>
using namespace std;

// A method to multiply two numbers using Russian Peasant method
unsigned int russianPeasant(unsigned int a, unsigned int b)
{
    int res = 0; // initialize result

    // While second number doesn't become 1
    while (b > 0)
    {
        // If second number becomes odd, add the first number to result
        if (b & 1)
            res = res + a;

        // Double the first number and halve the second number
        a = a << 1;
        b = b >> 1;
    }
    return res;
}

// Driver program to test above function
int main()
{
    cout << russianPeasant(18, 1) << endl;
    cout << russianPeasant(20, 12) << endl;
    return 0;
}
```

[Run on IDE](#)

Output:

18
240

How does this work?

The value of $a \times b$ is same as $(a \times 2)^*(b/2)$ if b is even, otherwise the value is same as $((a \times 2)^*(b/2) + a)$. In the while loop, we keep multiplying 'a' with 2 and keep dividing 'b' by 2. If 'b' becomes odd in loop, we add 'a' to 'res'. When value of 'b' becomes 1, the value of 'res' + 'a', gives us the result.

Note that when 'b' is a power of 2, the 'res' would remain 0 and 'a' would have the multiplication. See the reference for more information.

Reference:

<http://mathforum.org/dr.math/faq/faq.peasant.html>

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



GATE CS Corner Company Wise Coding Practice

Mathematical MathematicalAlgo

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.1

Average Difficulty : **2.1/5.0**
Based on **6** vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count all possible groups of size 2 or 3 that have sum as multiple of 3

Given an unsorted integer (positive values only) array of size 'n', we can form a group of two or three, the group should be such that the sum of all elements in that group is a multiple of 3. Count all possible number of groups that can be generated in this way.

```
Input: arr[] = {3, 6, 7, 2, 9}
Output: 8
// Groups are {3,6}, {3,9}, {9,6}, {7,2}, {3,6,9},
//           {3,7,2}, {7,2,6}, {7,2,9}
```

```
Input: arr[] = {2, 1, 3, 4}
Output: 4
// Groups are {2,1}, {2,4}, {2,1,3}, {2,4,3}
```

We strongly recommend that you click here and practice it, before moving on to the solution.

The idea is to see remainder of every element when divided by 3. A set of elements can form a group only if sun of their remainders is multiple of 3. Since the task is to enumerate groups, we count all elements with different remainders.

1. Hash all elements in a count array based on remainder, i.e,
for all elements $a[i]$, do $c[a[i]\%3]++$;
2. Now $c[0]$ contains the number of elements which when divided by 3 leave remainder 0 and similarly $c[1]$ for remainder 1 and $c[2]$ for 2.
3. Now for group of 2, we have 2 possibilities
 - a. 2 elements of remainder 0 group. Such possibilities are $c[0]*(c[0]-1)/2$
 - b. 1 element of remainder 1 and 1 from remainder 2 group
Such groups are $c[1]*c[2]$.
4. Now for group of 3,we have 4 possibilities
 - a. 3 elements from remainder group 0.
No. of such groups are $c[0]C3$
 - b. 3 elements from remainder group 1.
No. of such groups are $c[1]C3$
 - c. 3 elements from remainder group 2.

No. of such groups are $c[2]C_3$
d. 1 element from each of 3 groups.
No. of such groups are $c[0]*c[1]*c[2]$.
5. Add all the groups in steps 3 and 4 to obtain the result.

```
#include<stdio.h>

// Returns count of all possible groups that can be formed from elements
// of a[].
int findgroups(int arr[], int n)
{
    // Create an array C[3] to store counts of elements with remainder
    // 0, 1 and 2. c[i] would store count of elements with remainder i
    int c[3] = {0}, i;

    int res = 0; // To store the result

    // Count elements with remainder 0, 1 and 2
    for (i=0; i<n; i++)
        c[arr[i]%3]++;

    // Case 3.a: Count groups of size 2 from 0 remainder elements
    res += ((c[0]*(c[0]-1))>>1);

    // Case 3.b: Count groups of size 2 with one element with 1
    // remainder and other with 2 remainder
    res += c[1] * c[2];

    // Case 4.a: Count groups of size 3 with all 0 remainder elements
    res += (c[0] * (c[0]-1) * (c[0]-2))/6;

    // Case 4.b: Count groups of size 3 with all 1 remainder elements
    res += (c[1] * (c[1]-1) * (c[1]-2))/6;

    // Case 4.c: Count groups of size 3 with all 2 remainder elements
    res += ((c[2]*(c[2]-1)*(c[2]-2))/6);

    // Case 4.c: Count groups of size 3 with different remainders
    res += c[0]*c[1]*c[2];

    // Return total count stored in res
    return res;
}

// Driver program to test above functions
int main()
{
    int arr[] = {3, 6, 7, 2, 9};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Required number of groups are %d\n", findgroups(arr,n));
    return 0;
}
```

[Run on IDE](#)

Java

```
class FindGroups
{
    // Returns count of all possible groups that can be formed from elements
    // of a[].

    int findgroups(int arr[], int n)
    {
        // Create an array C[3] to store counts of elements with remainder
```

```

// 0, 1 and 2. c[i] would store count of elements with remainder i
int c[] = new int[]{0, 0, 0};
int i;

int res = 0; // To store the result

// Count elements with remainder 0, 1 and 2
for (i = 0; i < n; i++)
    c[arr[i] % 3]++;

// Case 3.a: Count groups of size 2 from 0 remainder elements
res += ((c[0] * (c[0] - 1)) >> 1);

// Case 3.b: Count groups of size 2 with one element with 1
// remainder and other with 2 remainder
res += c[1] * c[2];

// Case 4.a: Count groups of size 3 with all 0 remainder elements
res += (c[0] * (c[0] - 1) * (c[0] - 2)) / 6;

// Case 4.b: Count groups of size 3 with all 1 remainder elements
res += (c[1] * (c[1] - 1) * (c[1] - 2)) / 6;

// Case 4.c: Count groups of size 3 with all 2 remainder elements
res += ((c[2] * (c[2] - 1) * (c[2] - 2)) / 6);

// Case 4.c: Count groups of size 3 with different remainders
res += c[0] * c[1] * c[2];

// Return total count stored in res
return res;
}

public static void main(String[] args)
{
    FindGroups groups = new FindGroups();
    int arr[] = {3, 6, 7, 2, 9};
    int n = arr.length;
    System.out.println("Required number of groups are "
        + groups.findgroups(arr, n));
}
}

```

Run on IDE

Output:

Required number of groups are 8

Time Complexity: O(n)

Auxiliary Space: O(1)

Asked in: Amazon

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



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Arrays

Related Posts:

- Maximum sum subarray removing at most one element
- Steps to return to {1, 2, ..n} with specified movements
- Position of an element after stable sort
- Move all negative elements to end in order with extra space allowed
- Minimize the sum of product of two arrays with permutations allowed
- Subarrays with distinct elements
- Sort an array according to absolute difference with given value
- Sum of all elements between k1'th and k2'th smallest elements

(Login to Rate and Mark)

3

Average Difficulty : 3/5.0
Based on 38 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

C Program for Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- 1) Only one disk can be moved at a time.
- 2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- 3) No disk may be placed on top of a smaller disk.

```
#include <stdio.h>

// C recursive function to solve tower of hanoi puzzle
void towerOfHanoi(int n, char fromrod, char torod, char auxrod)
{
    if (n == 1)
    {
        printf("\n Move disk 1 from rod %c to rod %c", fromrod, torod);
        return;
    }
    towerOfHanoi(n-1, fromrod, auxrod, torod);
    printf("\n Move disk %d from rod %c to rod %c", n, fromrod, torod);
    towerOfHanoi(n-1, auxrod, torod, fromrod);
}

int main()
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    return 0;
}
```

[Run on IDE](#)

Output:

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

For n disks, total $2^n - 1$ moves are required.

Related Articles

- Recursive Functions
- Iterative solution to TOH puzzle
- Quiz on Recursion

References:

http://en.wikipedia.org/wiki/Tower_of_Hanoi

GATE CS Corner



Download

Free Download

www.unzipper.com



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Programs

([Login](#) to Rate and Mark)

Average Rating : **5/5.0**

Based on 1 vote(s)



Average Difficulty : **2.9/5.0**

2.9

Based on 17 vote(s)



Add to TODO List

Mark as DONE

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Horner's Method for Polynomial Evaluation

Given a polynomial of the form $c_nx^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x + c_0$ and a value of x , find the value of polynomial for a given value of x . Here c_n, c_{n-1}, \dots are integers (may be negative) and n is a positive integer.

Input is in the form of an array say $\text{poly}[]$ where $\text{poly}[0]$ represents coefficient for x^n and $\text{poly}[1]$ represents coefficient for x^{n-1} and so on.

Examples:

```
// Evaluate value of 2x3 - 6x2 + 2x - 1 for x = 3
Input: poly[] = {2, -6, 2, -1}, x = 3
Output: 5
```

```
// Evaluate value of 2x3 + 3x + 1 for x = 2
Input: poly[] = {2, 0, 3, 1}, x = 2
Output: 23
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A naive way to evaluate a polynomial is to one by one evaluate all terms. First calculate x^n , multiply the value with c_n , repeat the same steps for other terms and return the sum. Time complexity of this approach is $O(n^2)$ if we use a simple loop for evaluation of x^n . Time complexity can be improved to $O(n\log n)$ if we use [O\(Logn\) approach for evaluation of \$x^n\$](#) .

Horner's method can be used to evaluate polynomial in $O(n)$ time. To understand the method, let us consider the example of $2x^3 - 6x^2 + 2x - 1$. The polynomial can be evaluated as $((2x - 6)x + 2)x - 1$. The idea is to initialize result as coefficient of x^n which is 2 in this case, repeatedly multiply result with x and add next coefficient to result. Finally return result.

Following is C++ implementation of Horner's Method.

```
#include <iostream>
using namespace std;

// returns value of poly[0]x(n-1) + poly[1]x(n-2) + ... + poly[n-1]
int horner(int poly[], int n, int x)
{
    int result = poly[0]; // Initialize result

    // Evaluate value of polynomial using Horner's method
    for (int i=1; i<n; i++)
        result = result*x + poly[i];
```

```

    return result;
}

// Driver program to test above function.
int main()
{
    // Let us evaluate value of 2x3 - 6x2 + 2x - 1 for x = 3
    int poly[] = {2, -6, 2, -1};
    int x = 3;
    int n = sizeof(poly)/sizeof(poly[0]);
    cout << "Value of polynomial is " << horner(poly, n, x);
    return 0;
}

```

[Run on IDE](#)

Output:

Value of polynomial is 5

Time Complexity: O(n)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alatile.com



GATE CS Corner Company Wise Coding Practice

[Mathematical](#)

[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n

- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2

Average Difficulty : 2/5.0
Based on 13 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)[GATE CS](#)[Placements](#)[GeeksQuiz](#)[Login/Register](#)

Count trailing zeroes in factorial of a number

Given an integer n, write a function that returns count of trailing zeroes in n!.

Examples:

Input: n = 5

Output: 1

Factorial of 5 is 20 which has one trailing 0.

Input: n = 20

Output: 4

Factorial of 20 is 2432902008176640000 which has 4 trailing zeroes.

Input: n = 100

Output: 24

We strongly recommend that you click here and practice it, before moving on to the solution.

A simple method is to first calculate factorial of n, then count trailing 0s in the result (We can count trailing 0s by repeatedly dividing the factorial by 10 till the remainder is 0).

The above method can cause overflow for a slightly bigger numbers as factorial of a number is a big number (See factorial of 20 given in above examples). The idea is to consider **prime factors** of a factorial n. A trailing zero is always produced by prime factors 2 and 5. If we can count the number of 5s and 2s, our task is done. Consider the following examples.

n = 5: There is one 5 and 3 2s in prime factors of 5! ($2 * 2 * 2 * 3 * 5$). So count of trailing 0s is 1.

n = 11: There are two 5s and three 2s in prime factors of 11! ($2^8 * 3^4 * 5^2 * 7$). So count of trailing 0s is 2.

We can easily observe that the number of 2s in prime factors is always more than or equal to the number of 5s. So if we count 5s in prime factors, we are done. *How to count total number of 5s in prime factors of n!*? A simple way is to calculate $\text{floor}(n/5)$. For example, 7! has one 5, 10! has two 5s. It is done yet, there is one more thing to consider. Numbers like 25, 125, etc have more than one 5. For example if we consider 28!, we get one extra 5 and number of 0s become 6. Handling this is simple, first divide n by 5 and remove all single 5s, then divide by 25 to remove extra 5s and so on. Following is the summarized formula for counting trailing 0s.

```
Trailing 0s in n! = Count of 5s in prime factors of n!
= floor(n/5) + floor(n/25) + floor(n/125) + ....
```

Following is C++ program based on above formula.

```
// C++ program to count trailing 0s in n!
#include <iostream>
using namespace std;

// Function to return trailing 0s in factorial of n
int findTrailingZeros(int n)
{
    // Initialize result
    int count = 0;

    // Keep dividing n by powers of 5 and update count
    for (int i=5; n/i>=1; i *= 5)
        count += n/i;

    return count;
}

// Driver program to test above function
int main()
{
    int n = 100;
    cout << "Count of trailing 0s in " << 100
        << "!" << " is " << findTrailingZeros(n);
    return 0;
}
```

[Run on IDE](#)

Output:

```
Count of trailing 0s in 100! is 24
```

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



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

[Mathematical](#)[factorial](#)[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2

Average Difficulty : **2/5.0**
Based on **25** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Program for nth Catalan Number

Catalan numbers are a sequence of natural numbers that occurs in many interesting counting problems like following.

1) Count the number of expressions containing n pairs of parentheses which are correctly matched. For n = 3, possible expressions are ((())), ()(), ()(), (())(), (())().

2) Count the number of possible Binary Search Trees with n keys (See [this](#))

3) Count the number of full binary trees (A rooted binary tree is full if every vertex has either two children or no children) with n+1 leaves.

See [this](#) for more applications.

The first few Catalan numbers for n = 0, 1, 2, 3, ... are **1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...**

Recursive Solution

Catalan numbers satisfy the following recursive formula.

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^n C_i C_{n-i} \quad \text{for } n \geq 0;$$

Following is the implementation of above recursive formula.

```
#include<iostream>
using namespace std;

// A recursive function to find nth catalan number
unsigned long int catalan(unsigned int n)
{
    // Base case
    if (n <= 1) return 1;

    // catalan(n) is sum of catalan(i)*catalan(n-i-1)
    unsigned long int res = 0;
    for (int i=0; i<n; i++)
        res += catalan(i)*catalan(n-i-1);

    return res;
}

// Driver program to test above function
int main()
{
    for (int i=0; i<10; i++)
        cout << catalan(i) << " ";
    return 0;
}
```

[Run on IDE](#)

Java

```
class CatalnNumber {

    // A recursive function to find nth catalan number

    int catalan(int n) {
        int res = 0;

        // Base case
        if (n <= 1) {
            return 1;
        }
        for (int i = 0; i < n; i++) {
            res += catalan(i) * catalan(n - i - 1);
        }
        return res;
    }

    public static void main(String[] args) {
        CatalnNumber cn = new CatalnNumber();
        for (int i = 0; i < 10; i++) {
            System.out.print(cn.catalan(i) + " ");
        }
    }
}
```

[Run on IDE](#)

Python

```
# A recursive function to find nth catalan number
def catalan(n):
    # Base Case
    if n <= 1 :
        return 1

    # Catalan(n) is the sum of catalan(i)*catalan(n-i-1)
    res = 0
    for i in range(n):
        res += catalan(i) * catalan(n-i-1)

    return res

# Driver Program to test above function
for i in range(10):
    print catalan(i),
# This code is contributed by Nikhil Kumar Singh (nickzuck_007)
```

[Run on IDE](#)

Output :

```
1 1 2 5 14 42 132 429 1430 4862
```

Time complexity of above implementation is equivalent to nth catalan number.

$$T(n) = \sum_{i=0}^{n-1} T(i) * T(n - i) \quad \text{for } n \geq 0;$$

The value of nth catalan number is exponential that makes the time complexity exponential.

Dynamic Programming Solution

We can observe that the above recursive implementation does a lot of repeated work (we can see the same by drawing recursion tree). Since there are overlapping subproblems, we can use dynamic programming for this. Following is a Dynamic programming based implementation in C++.

```
#include<iostream>
using namespace std;

// A dynamic programming based function to find nth
// Catalan number
unsigned long int catalanDP(unsigned int n)
{
    // Table to store results of subproblems
    unsigned long int catalan[n+1];

    // Initialize first two values in table
    catalan[0] = catalan[1] = 1;

    // Fill entries in catalan[] using recursive formula
    for (int i=2; i<=n; i++)
    {
        catalan[i] = 0;
        for (int j=0; j<i; j++)
            catalan[i] += catalan[j] * catalan[i-j-1];
    }

    // Return last entry
    return catalan[n];
}

// Driver program to test above function
int main()
{
    for (int i = 0; i < 10; i++)
        cout << catalanDP(i) << " ";
    return 0;
}
```

[Run on IDE](#)

Output:

```
1 1 2 5 14 42 132 429 1430 4862
```

Time Complexity: Time complexity of above implementation is $O(n^2)$

Using Binomial Coefficient

We can also use the below formula to find nth catalan number in $O(n)$ time.

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

We have discussed a $O(n)$ approach to find binomial coefficient nCr .

```
#include<iostream>
using namespace std;

// Returns value of Binomial Coefficient C(n, k)
unsigned long int binomialCoeff(unsigned int n, unsigned int k)
{
    unsigned long int res = 1;
```

```

// Since C(n, k) = C(n, n-k)
if (k > n - k)
    k = n - k;

// Calculate value of [n*(n-1)*---*(n-k+1)] / [k*(k-1)*---*1]
for (int i = 0; i < k; ++i)
{
    res *= (n - i);
    res /= (i + 1);
}

return res;
}

// A Binomial coefficient based function to find nth catalan
// number in O(n) time
unsigned long int catalan(unsigned int n)
{
    // Calculate value of 2nCn
    unsigned long int c = binomialCoeff(2*n, n);

    // return 2nCn/(n+1)
    return c/(n+1);
}

// Driver program to test above functions
int main()
{
    for (int i = 0; i < 10; i++)
        cout << catalan(i) << " ";
    return 0;
}

```

[Run on IDE](#)

Output:

```
1 1 2 5 14 42 132 429 1430 4862
```

Time Complexity: Time complexity of above implementation is O(n).

We can also use below formula to find nth catalan number in O(n) time.

$$C_n = \frac{(2n)!}{(n+1)! n!} = \prod_{k=2}^n \frac{n+k}{k} \quad \text{for } n \geq 0$$

References:

http://en.wikipedia.org/wiki/Catalan_number

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Ali Tile BY **LOLA GROUP**

LOLA CERAMICS (M) SDN. BHD.
ADDRESS: 38, Jalan PJD 343, Taman Durianbaru, 4719 Petaling Jaya, Malaysia
TEL: +603-9066 2910 WEBSITE: www.alatile.com

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alatile.com

GATE CS Corner Company Wise Coding Practice

Dynamic Programming Mathematical catalan MathematicalAlgo series

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.3

Average Difficulty : **2.3/5.0**
Based on **29** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Write a function that generates one of 3 numbers according to given probabilities

You are given a function `rand(a, b)` which generates equiprobable random numbers between $[a, b]$ inclusive. Generate 3 numbers x, y, z with probability $P(x), P(y), P(z)$ such that $P(x) + P(y) + P(z) = 1$ using the given `rand(a,b)` function.

The idea is to utilize the equiprobable feature of the `rand(a,b)` provided. **Let the given probabilities be in percentage form, for example $P(x)=40\%$, $P(y)=25\%$, $P(z)=35\%$..**

Following are the detailed steps.

1) Generate a random number between 1 and 100. Since they are equiprobable, the probability of each number appearing is $1/100$.

2) Following are some important points to note about generated random number 'r'.

a) 'r' is smaller than or equal to $P(x)$ with probability $P(x)/100$.

b) 'r' is greater than $P(x)$ and smaller than or equal $P(x) + P(y)$ with $P(y)/100$.

c) 'r' is greater than $P(x) + P(y)$ and smaller than or equal 100 (or $P(x) + P(y) + P(z)$) with probability $P(z)/100$.

```
// This function generates 'x' with probability px/100, 'y' with
// probability py/100 and 'z' with probability pz/100:
// Assumption: px + py + pz = 100 where px, py and pz lie
// between 0 to 100
int random(int x, int y, int z, int px, int py, int pz)
{
    // Generate a number from 1 to 100
    int r = rand(1, 100);

    // r is smaller than px with probability px/100
    if (r <= px)
        return x;

    // r is greater than px and smaller than or equal to px+py
    // with probability py/100
    if (r <= (px+py))
        return y;

    // r is greater than px+py and smaller than or equal to 100
    // with probability pz/100
    else
        return z;
}
```

[Run on IDE](#)

This function will solve the purpose of generating 3 numbers with given three probabilities.

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

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [Randomized](#) [MathematicalAlgo](#)

Related Posts:

- [Nth character in Concatenated Decimal String](#)
- [Find the highest occurring digit in prime numbers in a range](#)
- [Smallest number to multiply to convert floating point to natural](#)
- [Generate all palindromic numbers less than n](#)
- [Number of sextuplets \(or six values\) that satisfy an equation](#)
- [Circular primes less than n](#)
- [Sphenic Number](#)
- [Minimum sum of two numbers formed from digits of an array](#)

(Login to Rate and Mark)

1.6 Average Difficulty : **1.6/5.0**
Based on **5** vote(s)

Add to TODO List
 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find Excel column name from a given column number

MS Excel columns has a pattern like A, B, C, ... ,Z, AA, AB, AC,.... ,AZ, BA, BB, ... ZZ, AAA, AAB etc. In other words, column 1 is named as "A", column 2 as "B", column 27 as "AA".

Given a column number, find its corresponding Excel column name. Following are more examples.

Input	Output
26	Z
51	AY
52	AZ
80	CB
676	YZ
702	ZZ
705	AAC

We strongly recommend that you click here and practice it, before moving on to the solution.

Thanks to [Mrigank Dembla](#) for suggesting the below solution in a comment.

Suppose we have a number n, let's say 28. so corresponding to it we need to print the column name. We need to take remainder with 26.

If remainder with 26 comes out to be 0 (meaning 26, 52 and so on) then we put 'Z' in the output string and new n becomes $n/26 - 1$ because here we are considering 26 to be 'Z' while in actual it's 25th with respect to 'A'.

Similarly if the remainder comes out to be non zero. (like 1, 2, 3 and so on) then we need to just insert the char accordingly in the string and do $n = n/26$.

Finally we reverse the string and print.

Example:

$n = 700$

Remainder ($n \% 26$) is 24. So we put 'X' in output string and n becomes $n/26$ which is 26.

Remainder ($26 \% 26$) is 0. So we put 'Z' in output string and n becomes $n/26 - 1$ which is 0.

Following is C++ implementation of above approach.

```
#include<bits/stdc++.h>
#define MAX 50
using namespace std;

// Function to print Excel column name for a given column number
void printString(int n)
{
    char str[MAX]; // To store result (Excel column name)
    int i = 0; // To store current index in str which is result

    while (n>0)
    {
        // Find remainder
        int rem = n%26;

        // If remainder is 0, then a 'Z' must be there in output
        if (rem==0)
        {
            str[i++] = 'Z';
            n = (n/26)-1;
        }
        else // If remainder is non-zero
        {
            str[i++] = (rem-1) + 'A';
            n = n/26;
        }
    }
    str[i] = '\0';

    // Reverse the string and print result
    reverse(str, str + strlen(str));
    cout << str << endl;

    return;
}

// Driver program to test above function
int main()
{
    printString(26);
    printString(51);
    printString(52);
    printString(80);
    printString(676);
    printString(702);
    printString(705);
    return 0;
}
```

Python

```
# Python program to find Excel column name from a
# given column number

MAX = 50

# Function to print Excel column name for a given column number
def printString(n):

    # To store result (Excel column name)
    string = ["\0"]*MAX

    # To store current index in str which is result
    i = 0

    while n > 0:
        # Find remainder
        rem = n%26

        # if remainder is 0, then a 'Z' must be there in output
        if rem == 0:
            string[i] = 'Z'
            i += 1
            n = (n/26)-1
        else:
            string[i] = chr((rem-1) + ord('A'))
            i += 1
            n = n/26
        string[i] = '\0'

    # Reverse the string and print result
    string = string[::-1]
    print "".join(string)

# Driver program to test the above Function
printString(26)
printString(51)
printString(52)
printString(80)
printString(676)
printString(702)
printString(705)

# This code is contributed by BHAVYA JAIN
```

Output

Z
AY
AZ
CB
YZ
ZZ
AAC

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



GATE CS Corner Company Wise Coding Practice

Strings

Related Posts:

- Nth character in Concatenated Decimal String
- Convert to a string that is repetition of a substring of k length
- Minimum characters to be added at front to make string palindrome
- Count All Palindrome Sub-Strings in a String
- Check for Palindrome after every character replacement Query
- Group all occurrences of characters according to first appearance
- Count characters at same position as in English alphabets
- Find if an array of strings can be chained to form a circle | Set 2

(Login to Rate and Mark)

2.8

Average Difficulty : 2.8/5.0
Based on 42 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find next greater number with same set of digits

Given a number n, find the smallest number that has same set of digits as n and is greater than n. If x is the greatest possible number with its set of digits, then print “not possible”.

Examples:

For simplicity of implementation, we have considered input number as a string.

```
Input: n = "218765"
Output: "251678"
```

```
Input: n = "1234"
Output: "1243"
```

```
Input: n = "4321"
Output: "Not Possible"
```

```
Input: n = "534976"
Output: "536479"
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Following are few observations about the next greater number.

- 1) If all digits sorted in descending order, then output is always “Not Possible”. For example, 4321.
- 2) If all digits are sorted in ascending order, then we need to swap last two digits. For example, 1234.
- 3) For other cases, we need to process the number from rightmost side (why? because we need to find the smallest of all greater numbers)

You can now try developing an algorithm yourself.

Following is the algorithm for finding the next greater number.

- I) Traverse the given number from rightmost digit, keep traversing till you find a digit which is smaller than the previously traversed digit. For example, if the input number is “534976”, we stop at **4** because 4 is smaller than next digit 9. If we do not find such a digit, then output is “Not Possible”.
- II) Now search the right side of above found digit ‘d’ for the smallest digit greater than ‘d’. For “53**4**976”, the right side of 4 contains “976”. The smallest digit greater than 4 is **6**.
- III) Swap the above found two digits, we get **536974** in above example.

IV) Now sort all digits from position next to 'd' to the end of number. The number that we get after sorting is the output. For above example, we sort digits in bold 536**974**. We get "536**479**" which is the next greater number for input 534976.

Following is C++ implementation of above approach.

```
// C++ program to find the smallest number which greater than a given number
// and has same set of digits as given number
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

// Utility function to swap two digits
void swap(char *a, char *b)
{
    char temp = *a;
    *a = *b;
    *b = temp;
}

// Given a number as a char array number[], this function finds the
// next greater number. It modifies the same array to store the result
void findNext(char number[], int n)
{
    int i, j;

    // I) Start from the right most digit and find the first digit that is
    // smaller than the digit next to it.
    for (i = n-1; i > 0; i--)
        if (number[i] > number[i-1])
            break;

    // If no such digit is found, then all digits are in descending order
    // means there cannot be a greater number with same set of digits
    if (i==0)
    {
        cout << "Next number is not possible";
        return;
    }

    // II) Find the smallest digit on right side of (i-1)'th digit that is
    // greater than number[i-1]
    int x = number[i-1], smallest = i;
    for (j = i+1; j < n; j++)
        if (number[j] > x && number[j] < number[smallest])
            smallest = j;

    // III) Swap the above found smallest digit with number[i-1]
    swap(&number[smallest], &number[i-1]);

    // IV) Sort the digits after (i-1) in ascending order
    sort(number + i, number + n);

    cout << "Next number with same set of digits is " << number;
    return;
}

// Driver program to test above function
int main()
{
    char digits[] = "534976";
    int n = strlen(digits);
    findNext(digits, n);
    return 0;
}
```

Run on IDE

Output:

Next number with same set of digits is 536479

The above implementation can be optimized in following ways.

1) We can use binary search in step II instead of linear search.

2) In step IV, instead of doing simple sort, we can apply some clever technique to do it in linear time. Hint: We know that all digits are linearly sorted in reverse order except one digit which was swapped.

With above optimizations, we can say that the time complexity of this method is O(n).

Asked in: **Adobe, Amazon, Hike, Microsoft, Morgan Stanley, Oxigen Wallet, Samsung, Snapdeal, Vizury Interactive S, Zillious**

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

GATE CS Corner Company Wise Coding Practice

Mathematical MathematicalAlgo number-digits

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

[\(Login to Rate and Mark\)](#)**3.3**Average Difficulty : **3.3/5.0**
Based on **74** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count Possible Decodings of a given Digit Sequence

Let 1 represent 'A', 2 represents 'B', etc. Given a digit sequence, count the number of possible decodings of the given digit sequence.

Examples:

```
Input: digits[] = "121"
Output: 3
// The possible decodings are "ABA", "AU", "LA"

Input: digits[] = "1234"
Output: 3
// The possible decodings are "ABCD", "LCD", "AWD"
```

An empty digit sequence is considered to have one decoding. It may be assumed that the input contains valid digits from 0 to 9 and there are no leading 0's, no extra trailing 0's and no two or more consecutive 0's.

We strongly recommend to minimize the browser and try this yourself first.

This problem is recursive and can be broken in sub-problems. We start from end of the given digit sequence. We initialize the total count of decodings as 0. We recur for two subproblems.

- 1) If the last digit is non-zero, recur for remaining (n-1) digits and add the result to total count.
- 2) If the last two digits form a valid character (or smaller than 27), recur for remaining (n-2) digits and add the result to total count.

Following is C++ implementation of the above approach.

```
// A naive recursive C++ implementation to count number of decodings
// that can be formed from a given digit sequence
#include <iostream>
#include <cstring>
using namespace std;

// Given a digit sequence of length n, returns count of possible
// decodings by replacing 1 with A, 2 with B, ... 26 with Z
int countDecoding(char *digits, int n)
{
    // base cases
    if (n == 0 || n == 1)
        return 1;

    int count = 0; // Initialize count

    // If the last digit is not 0, then last digit must add to
    // the number of words
    if (digits[n-1] > '0')
        count = countDecoding(digits, n-1);

    // If the last digit is 0, then it must be combined with
    // previous digit to make a valid character
    if (n > 1 && digits[n-2] >='1' && digits[n-2] <='9' && digits[n-1] >='0' && digits[n-1] <='3')
        count += countDecoding(digits, n-2);
}
```

```

// If the last two digits form a number smaller than or equal to 26,
// then consider last two digits and recur
if (digits[n-2] < '2' || (digits[n-2] == '2' && digits[n-1] < '7') )
    count += countDecoding(digits, n-2);

return count;
}

// Driver program to test above function
int main()
{
    char digits[] = "1234";
    int n = strlen(digits);
    cout << "Count is " << countDecoding(digits, n);
    return 0;
}

```

[Run on IDE](#)

Output:

```
Count is 3
```

The time complexity of above the code is exponential. If we take a closer look at the above program, we can observe that the recursive solution is similar to [Fibonacci Numbers](#). Therefore, we can optimize the above solution to work in $O(n)$ time using [Dynamic Programming](#). Following is C++ implementation for the same.

```

// A Dynamic Programming based C++ implementation to count decodings
#include <iostream>
#include <cstring>
using namespace std;

// A Dynamic Programming based function to count decodings
int countDecodingDP(char *digits, int n)
{
    int count[n+1]; // A table to store results of subproblems
    count[0] = 1;
    count[1] = 1;

    for (int i = 2; i <= n; i++)
    {
        count[i] = 0;

        // If the last digit is not 0, then last digit must add to
        // the number of words
        if (digits[i-1] > '0')
            count[i] = count[i-1];

        // If second last digit is smaller than 2 and last digit is
        // smaller than 7, then last two digits form a valid character
        if (digits[i-2] < '2' || (digits[i-2] == '2' && digits[i-1] < '7') )
            count[i] += count[i-2];
    }
    return count[n];
}

// Driver program to test above function
int main()
{
    char digits[] = "1234";
    int n = strlen(digits);
    cout << "Count is " << countDecodingDP(digits, n);
    return 0;
}

```

[Run on IDE](#)

Output:

Count is 3

Time Complexity of the above solution is $O(n)$ and it requires $O(n)$ auxiliary space. We can reduce auxiliary space to $O(1)$ by using space optimized version discussed in the [Fibonacci Number Post](#).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [Fibonacci](#) [MathematicalAlgo](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 36 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Calculate the angle between hour hand and minute hand

This problem is known as [Clock angle problem](#) where we need to find angle between hands of an analog clock at a given time.

Examples:

Input: h = 12:00, m = 30.00
Output: 165 degree

Input: h = 3.00, m = 30.00
Output: 75 degree

We strongly recommend that you click here and practice it, before moving on to the solution.

The idea is to take 12:00 (h = 12, m = 0) as a reference. Following are detailed steps.

- 1) Calculate the angle made by hour hand with respect to 12:00 in h hours and m minutes.
- 2) Calculate the angle made by minute hand with respect to 12:00 in h hours and m minutes.
- 3) The difference between two angles is the angle between two hands.

How to calculate the two angles with respect to 12:00?

The minute hand moves 360 degree in 60 minute(or 6 degree in one minute) and hour hand moves 360 degree in 12 hours(or 0.5 degree in 1 minute). In h hours and m minutes, the minute hand would move $(h*60 + m)*6$ and hour hand would move $(h*60 + m)*0.5$.

```
// C program to find angle between hour and minute hands
#include <stdio.h>
#include <stdlib.h>

// Utility function to find minimum of two integers
int min(int x, int y) { return (x < y)? x: y; }

int calcAngle(double h, double m)
{
    // validate the input
    if (h <0 || m < 0 || h >12 || m > 60)
        printf("Wrong input");

    if (h == 12) h = 0;
    if (m == 60) m = 0;

    // calculate the angle
    double angle = (h*30 + m/2) - (m*6);
    if (angle < 0)
        angle += 360;
}
```

```

// Calculate the angles moved by hour and minute hands
// with reference to 12:00
int hour_angle = 0.5 * (h*60 + m);
int minute_angle = 6*m;

// Find the difference between two angles
int angle = abs(hour_angle - minute_angle);

// Return the smaller angle of two possible angles
angle = min(360-angle, angle);

return angle;
}

// Driver program to test above function
int main()
{
    printf("%d \n", calcAngle(9, 60));
    printf("%d \n", calcAngle(3, 30));
    return 0;
}

```

[Run on IDE](#)

Output:

```

90
75

```

Asked in: Amazon, Infinera, Paytm, Sales Force

Exercise: Find all times when hour and minute hands get superimposed.

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



GATE CS Corner Company Wise Coding Practice

[Mathematical](#)
[MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 28 vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count number of binary strings without consecutive 1's

Given a positive integer N, count all possible distinct binary strings of length N such that there are no consecutive 1's.

Examples:

```
Input: N = 2
Output: 3
// The 3 strings are 00, 01, 10
```

```
Input: N = 3
Output: 5
// The 5 strings are 000, 001, 010, 100, 101
```

We strongly recommend that you click here and practice it, before moving on to the solution.

This problem can be solved using Dynamic Programming. Let $a[i]$ be the number of binary strings of length i which do not contain any two consecutive 1's and which end in 0. Similarly, let $b[i]$ be the number of such strings which end in 1. We can append either 0 or 1 to a string ending in 0, but we can only append 0 to a string ending in 1. This yields the recurrence relation:

```
a[i] = a[i - 1] + b[i - 1]
b[i] = a[i - 1]
```

The base cases of above recurrence are $a[1] = b[1] = 1$. The total number of strings of length i is just $a[i] + b[i]$.

Following is the implementation of above solution. In the following implementation, indexes start from 0. So $a[i]$ represents the number of binary strings for input length $i+1$. Similarly, $b[i]$ represents binary strings for input length $i+1$.

```
// C++ program to count all distinct binary strings
// without two consecutive 1's
#include <iostream>
using namespace std;
```

```

int countStrings(int n)
{
    int a[n], b[n];
    a[0] = b[0] = 1;
    for (int i = 1; i < n; i++)
    {
        a[i] = a[i-1] + b[i-1];
        b[i] = a[i-1];
    }
    return a[n-1] + b[n-1];
}

// Driver program to test above functions
int main()
{
    cout << countStrings(3) << endl;
    return 0;
}

```

[Run on IDE](#)

Java

```

class Subset_sum
{
    static int countStrings(int n)
    {
        int a[] = new int [n];
        int b[] = new int [n];
        a[0] = b[0] = 1;
        for (int i = 1; i < n; i++)
        {
            a[i] = a[i-1] + b[i-1];
            b[i] = a[i-1];
        }
        return a[n-1] + b[n-1];
    }
    /* Driver program to test above function */
    public static void main (String args[])
    {
        System.out.println(countStrings(3));
    }
}/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

5

Source:

courses.csail.mit.edu/6.006/oldquizzes/solutions/q2-f2009-sol.pdf

If we take a closer look at the pattern, we can observe that the count is actually $(n+2)$ 'th Fibonacci number for $n \geq 1$. The **Fibonacci Numbers** are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 141,

```

n = 1, count = 2 = fib(3)
n = 2, count = 3 = fib(4)
n = 3, count = 5 = fib(5)
n = 4, count = 8 = fib(6)
n = 5, count = 13 = fib(7)
.....

```

Therefore we can count the strings in O(Log n) time also using the method 5 [here](#).

Asked in: **Flipkart, Microsoft, Snapdeal**

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



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [binary-string](#) [Dynamic Programming](#) [Fibonacci](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 50 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find the smallest number whose digits multiply to a given number n

Given a number 'n', find the smallest number 'p' such that if we multiply all digits of 'p', we get 'n'. The result 'p' should have minimum two digits.

Examples:

```
Input: n = 36
Output: p = 49
// Note that 4*9 = 36 and 49 is the smallest such number
```

```
Input: n = 100
Output: p = 455
// Note that 4*5*5 = 100 and 455 is the smallest such number
```

```
Input: n = 1
Output: p = 11
// Note that 1*1 = 1
```

```
Input: n = 13
Output: Not Possible
```

For a given n, following are the two cases to be considered.

Case 1: n < 10 When n is smaller than 9, the output is always n+10. For example for n = 7, output is 17. For n = 9, output is 19.

Case 2: n >= 10 Find all factors of n which are between 2 and 9 (both inclusive). The idea is to start searching from 9 so that the number of digits in result are minimized. For example 9 is preferred over 33 and 8 is preferred over 24.

Store all found factors in an array. The array would contain digits in non-increasing order, so finally print the array in reverse order.

Following is C implementation of above concept.

```
#include<stdio.h>

// Maximum number of digits in output
#define MAX 50

// prints the smallest number whose digits multiply to n
void findSmallest(int n)
{
    int i, j=0;
    int res[MAX]; // To store digits of result in reverse order
```

```

// Case 1: If number is smaller than 10
if (n < 10)
{
    printf("%d", n+10);
    return;
}

// Case 2: Start with 9 and try every possible digit
for (i=9; i>1; i--)
{
    // If current digit divides n, then store all
    // occurrences of current digit in res
    while (n%i == 0)
    {
        n = n/i;
        res[j] = i;
        j++;
    }
}

// If n could not be broken in form of digits (prime factors of n
// are greater than 9)
if (n > 10)
{
    printf("Not possible");
    return;
}

// Print the result array in reverse order
for (i=j-1; i>=0; i--)
    printf("%d", res[i]);
}

// Driver program to test above function
int main()
{
    findSmallest(7);
    printf("\n");

    findSmallest(36);
    printf("\n");

    findSmallest(13);
    printf("\n");

    findSmallest(100);
    return 0;
}

```

Run on IDE

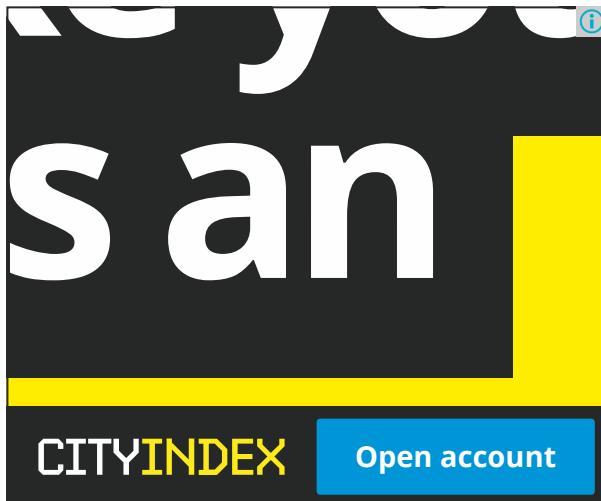
Output:

```

17
49
Not possible
455

```

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



GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#) [number-digits](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

3

Average Difficulty : 3/5.0
Based on 11 vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Draw a circle without floating point arithmetic

Given a radius of a circle, draw the circle without using floating point arithmetic.

Following program uses a simple concept. Let the radius of the circle be r. Consider a square of size $(2r+1) \times (2r+1)$ around the circle to be drawn. Now walk through every point inside the square. For every every point (x,y) , if (x, y) lies inside the circle ($x^2 + y^2 < r^2$), then print it, otherwise print space.

```
#include <stdio.h>

void drawCircle(int r)
{
    // Consider a rectangle of size N*N
    int N = 2*r+1;

    int x, y; // Coordinates inside the rectangle

    // Draw a square of size N*N.
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            // Start from the left most corner point
            x = i-r;
            y = j-r;

            // If this point is inside the circle, print it
            if (x*x + y*y <= r*r+1 )
                printf(".");
            else // If outside the circle, print space
                printf(" ");
            printf(" ");
        }
        printf("\n");
    }

    // Driver Program to test above function
    int main()
    {
        drawCircle(8);
        return 0;
    }
}
```

[Run on IDE](#)

Output:



Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner

An advertisement for Airbnb. It features the Airbnb logo at the top left. Below it, the text "Live There with Airbnb" is displayed in large white letters. Underneath that, a "Sign up now" button is visible. The main body of the ad contains the text "From bedrooms to apartments to villas - Airbnb has it all." followed by a large circular arrow icon pointing right. At the bottom left, the website "airbnb.com" is listed.

See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Articles

(Login to Rate and Mark)

Average Rating : **3/5.0**

Average Difficulty : **3.2/5.0**

3.2

Based on **4** vote(s)



Add to TODO List



Mark as DONE

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

How to check if an instance of 8 puzzle is solvable?

What is 8 puzzle?

Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles in order using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

1	2	3
4	5	6
7	8	

Goal State

Empty space can be anywhere

How to find if given state is solvable?

Following are two examples, the first example can reach goal state by a series of slides. The second example cannot.

1	8	2
	4	3
7	6	5

Given State

Solvable

We can reach goal state by sliding tiles using blank space.

8	1	2
	4	3
7	6	5

Given State

Not Solvable

We can not reach goal state by sliding tiles using blank space.

Following is simple rule to check if a 8 puzzle is solvable.

It is not possible to solve an instance of 8 puzzle if number of inversions is odd in the input state. In the examples given in above figure, the first example has 10 inversions, therefore solvable. The second example has 11 inversions, therefore unsolvable.

What is inversion?

A pair of tiles form an inversion if the values on tiles are in reverse order of their appearance in goal state. For

example, the following instance of 8 puzzle has two inversions, (8, 6) and (8, 7).

```

1   2   3
4   _   5
8   6   7

```

Following is a simple C++ program to check whether a given instance of 8 puzzle is solvable or not. The idea is simple, we count inversions in the given 8 puzzle.

```

// C++ program to check if a given instance of 8 puzzle is solvable or not
#include <iostream>
using namespace std;

// A utility function to count inversions in given array 'arr[]'
int getInvCount(int arr[])
{
    int inv_count = 0;
    for (int i = 0; i < 9 - 1; i++)
        for (int j = i+1; j < 9; j++)
            // Value 0 is used for empty space
            if (arr[j] && arr[i] && arr[i] > arr[j])
                inv_count++;
    return inv_count;
}

// This function returns true if given 8 puzzle is solvable.
bool isSolvable(int puzzle[3][3])
{
    // Count inversions in given 8 puzzle
    int invCount = getInvCount((int *)puzzle);

    // return true if inversion count is even.
    return (invCount%2 == 0);
}

/* Driver progra to test above functions */
int main(int argc, int** args)
{
    int puzzle[3][3] = {{1, 8, 2},
                        {0, 4, 3}, // Value 0 is used for empty space
                        {7, 6, 5}};
    isSolvable(puzzle)? cout << "Solvable":
                           cout << "Not Solvable";
    return 0;
}

```

[Run on IDE](#)

Output:

```
Solvable
```

Note that the above implementation uses simple algorithm for inversion count. It is done this way for simplicity. The code can be optimized to O(nLogn) using the [merge sort based algorithm for inversion count](#).

How does this work?

The idea is based on the fact the parity of inversions remains same after a set of moves, i.e., if the inversion count is odd in initial stage, then it remain odd after any sequence of moves and if the inversion count is even, then it remains even after any sequence of moves. In the goal state, there are 0 inversions. So we can reach goal state only from a state which has even inversion count.

How parity of inversion count is invariant?

When we slide a tile, we either make a row move (moving a left or right tile into the blank space), or make a

column move (moving a up or down tile to the blank space).

a) A row move doesn't change the inversion count. See following example

1	2	3	Row Move	1	2	3
4	_	5	----->	_	4	5
8	6	7		8	6	7

Inversion count remains 2 after the move

1	2	3	Row Move	1	2	3
4	_	5	----->	4	5	_
8	6	7		8	6	7

Inversion count remains 2 after the move

b) A column move does one of the following three.

.....(i) Increases inversion count by 2. See following example.

1	2	3	Column Move	1	_	3
4	_	5	----->	4	2	5
8	6	7		8	6	7

Inversion count increases by 2 (changes from 2 to 4)

.....(ii) Decreases inversion count by 2

1	3	4	Column Move	1	3	4
5	_	6	----->	5	2	6
7	2	8		7	_	8

Inversion count decreases by 2 (changes from 5 to 3)

.....(iii) Keeps the inversion count same.

1	2	3	Column Move	1	2	3
4	_	5	----->	4	6	5
7	6	8		7	_	8

Inversion count remains 1 after the move

So if a move either increases/decreases inversion count by 2, or keeps the inversion count same, then it is not possible to change parity of a state by any sequence of row/column moves.

Exercise: How to check if a given instance of 15 puzzle is solvable or not. In a 15 puzzle, we have 4×4 board where 15 tiles have a number and one empty space. Note that the above simple rules of inversion count don't directly work for 15 puzzle, the rules need to be modified for 15 puzzle.

Related Article:

[How to check if an instance of 15 puzzle is solvable?](#)

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

Ali Tile BY **LOLA GROUP**

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

4.4

Average Difficulty : **4.4/5.0**
Based on **5** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Birthday Paradox

How many people must be there in a room to make the probability 100% that two people in the room have same birthday?

Answer: 367 (since there are 366 possible birthdays, including February 29).

The above question was simple. Try the below question yourself.

How many people must be there in a room to make the probability 50% that two people in the room have same birthday?

Answer: 23

The number is surprisingly very low. In fact, we need only 70 people to make the probability 99.9 %.

Let us discuss the generalized formula.

What is the probability that two persons among n have same birthday?

Let the probability that two people in a room with n have same birthday be P(same). P(Same) can be easily evaluated in terms of P(different) where P(different) is the probability that all of them have different birthday.

$$P(\text{same}) = 1 - P(\text{different})$$

P(different) can be written as $1 \times (364/365) \times (363/365) \times (362/365) \times \dots \times (1 - (n-1)/365)$

How did we get the above expression?

Persons from first to last can get birthdays in following order for all birthdays to be distinct:

The first person can have any birthday among 365

The second person should have a birthday which is not same as first person

The third person should have a birthday which is not same as first two persons.

.....

.....

The n'th person should have a birthday which is not same as any of the earlier considered (n-1) persons.

Approximation of above expression

The above expression can be approximated using Taylor's Series.

$$e^x = 1 + x + \frac{x^2}{2!} + \dots$$

$$e^x \approx 1 + x.$$

provides a first-order approximation for ex for x << 1:

To apply this approximation to the first expression derived for p(different), set x = -a / 365. Thus,

$$e^{-a/365} \approx 1 - \frac{a}{365}.$$

The above expression derived for p(different) can be written as

$$1 \times (1 - 1/365) \times (1 - 2/365) \times (1 - 3/365) \times \dots \times (1 - (n-1)/365)$$

By putting the value of $1 - a/365$ as $e^{-a/365}$, we get following.

$$\begin{aligned} &\approx 1 \times e^{-1/365} \times e^{-2/365} \dots e^{-(n-1)/365} \\ &= 1 \times e^{-(1+2+\dots+(n-1))/365} \\ &= e^{-(n(n-1)/2)/365}. \end{aligned}$$

Therefore,

$$p(\text{same}) = 1 - p(\text{different}) \approx 1 - e^{-n(n-1)/(2 \times 365)}.$$

An even coarser approximation is given by

$$p(\text{same}) \approx 1 - e^{-n^2/(2 \times 365)},$$

By taking Log on both sides, we get the reverse formula.

$$n \approx \sqrt{2 \times 365 \ln\left(\frac{1}{1-p(\text{same})}\right)}.$$

Using the above approximate formula, we can approximate number of people for a given probability. For example the following C++ function find() returns the smallest n for which the probability is greater than the given p.

C++ Implementation of approximate formula.

The following is C++ program to approximate number of people for a given probability.

```
// C++ program to approximate number of people in Birthday Paradox
// problem
#include <cmath>
#include <iostream>
using namespace std;

// Returns approximate number of people for a given probability
int find(double p)
{
    return ceil(sqrt(2*365*log(1/(1-p))));
}

int main()
{
    cout << find(0.70);
}
```

[Run on IDE](#)

Output:

30

Source:

http://en.wikipedia.org/wiki/Birthday_problem

Applications:

- 1) Birthday Paradox is generally discussed with hashing to show importance of collision handling even for a small

set of keys.

2) Birthday Attack

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



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [Randomized](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

3.5

Average Difficulty : **3.5/5.0**
Based on **18** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Multiply two polynomials

Given two polynomials represented by two arrays, write a function that multiplies given two polynomials.

Example:

```
Input: A[] = {5, 0, 10, 6}
       B[] = {1, 2, 4}
Output: prod[] = {5, 10, 30, 26, 52, 24}
```

The first input array represents "5 + 0x¹ + 10x² + 6x³"

The second array represents "1 + 2x¹ + 4x²"

And Output is "5 + 10x¹ + 30x² + 26x³ + 52x⁴ + 24x⁵"

We strongly recommend that you click here and practice it, before moving on to the solution.

A simple solution is to one by one consider every term of first polynomial and multiply it with every term of second polynomial. Following is algorithm of this simple method.

```
multiply(A[0..m-1], B[0..n-1])
1) Create a product array prod[] of size m+n-1.
2) Initialize all entries in prod[] as 0.
3) Travers array A[] and do following for every element A[i]
... (3.a) Traverse array B[] and do following for every element B[j]
        prod[i+j] = prod[i+j] + A[i] * B[j]
4) Return prod[].
```

The following is C++ implementation of above algorithm.

```
// Simple C++ program to multiply two polynomials
#include <iostream>
using namespace std;

// A[] represents coefficients of first polynomial
// B[] represents coefficients of second polynomial
// m and n are sizes of A[] and B[] respectively
int *multiply(int A[], int B[], int m, int n)
{
    int *prod = new int[m+n-1];

    // Initialize the product polynomial
    for (int i = 0; i < m+n-1; i++)
        prod[i] = 0;
```

```

prod[i] = 0;

// Multiply two polynomials term by term

// Take ever term of first polynomial
for (int i=0; i<m; i++)
{
    // Multiply the current term of first polynomial
    // with every term of second polynomial.
    for (int j=0; j<n; j++)
        prod[i+j] += A[i]*B[j];
}

return prod;
}

// A utility function to print a polynomial
void printPoly(int poly[], int n)
{
    for (int i=0; i<n; i++)
    {
        cout << poly[i];
        if (i != 0)
            cout << "x^" << i ;
        if (i != n-1)
            cout << " + ";
    }
}

// Driver program to test above functions
int main()
{
    // The following array represents polynomial 5 + 10x^2 + 6x^3
    int A[] = {5, 0, 10, 6};

    // The following array represents polynomial 1 + 2x + 4x^2
    int B[] = {1, 2, 4};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);

    cout << "First polynomial is \n";
    printPoly(A, m);
    cout << "\nSecond polynomial is \n";
    printPoly(B, n);

    int *prod = multiply(A, B, m, n);

    cout << "\nProduct polynomial is \n";
    printPoly(prod, m+n-1);

    return 0;
}

```

[Run on IDE](#)

Output

```

First polynomial is
5 + 0x^1 + 10x^2 + 6x^3
Second polynomial is
1 + 2x^1 + 4x^2
Product polynomial is
5 + 10x^1 + 30x^2 + 26x^3 + 52x^4 + 24x^5

```

Time complexity of the above solution is $O(mn)$. If size of two polynomials same, then time complexity is $O(n^2)$.

Can we do better?

There are methods to do multiplication faster than $O(n^2)$ time. These methods are mainly based on **divide and**

conquer. Following is one simple method that divides the given polynomial (of degree n) into two polynomials one containing lower degree terms(lower than $n/2$) and other containing higher degree terms (higher than or equal to $n/2$)

Let the two given polynomials be A and B.

For simplicity, Let us assume that the given two polynomials are of same degree and have degree in powers of 2, i.e., $n = 2^i$

The polynomial 'A' can be written as $A_0 + A_1*x^{n/2}$

The polynomial 'B' can be written as $B_0 + B_1*x^{n/2}$

For example $1 + 10x + 6x^2 - 4x^3 + 5x^4$ can be written as $(1 + 10x) + (6 - 4x + 5x^2)*x^2$

$$\begin{aligned} A * B &= (A_0 + A_1*x^{n/2}) * (B_0 + B_1*x^{n/2}) \\ &= A_0*B_0 + A_0*B_1*x^{n/2} + A_1*B_0*x^{n/2} + A_1*B_1*x^n \\ &= A_0*B_0 + (A_0*B_1 + A_1*B_0)x^{n/2} + A_1*B_1*x^n \end{aligned}$$

So the above divide and conquer approach requires 4 multiplications and $O(n)$ time to add all 4 results. Therefore the time complexity is $T(n) = 4T(n/2) + O(n)$. The solution of the recurrence is $O(n^2)$ which is same as the above simple solution.

The idea is to reduce number of multiplications to 3 and make the recurrence as $T(n) = 3T(n/2) + O(n)$

How to reduce number of multiplications?

This requires a little trick similar to [Strassen's Matrix Multiplication](#). We do following 3 multiplications.

```
X = (A0 + A1)*(B0 + B1) // First Multiplication
Y = A0*B0 // Second
Z = A1*B1 // Third
```

The missing middle term in above multiplication equation $A_0*B_0 + (A_0*B_1 + A_1*B_0)x^{n/2} + A_1*B_1*x^n$ can be obtained using below.

$$A_0*B_1 + A_1*B_0 = X - Y - Z$$

So the time taken by this algorithm is $T(n) = 3T(n/2) + O(n)$

The solution of above recurrence is $O(n^{\lg 3})$ which is better than $O(n^2)$.

We will soon be discussing implementation of above approach.

There is a $O(n\log n)$ algorithm also that uses Fast Fourier Transform to multiply two polynomials (Refer [this](#) and [this](#) for details)

Sources:

<http://www.cse.ust.hk/~dehai/271/notes/L03/L03.pdf>

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



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Divide and Conquer Mathematical Divide and Conquer

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 11 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count Distinct Non-Negative Integer Pairs (x, y) that Satisfy the Inequality $x^*x + y^*y < n$

Given a positive number n, count all distinct Non-Negative Integer pairs (x, y) that satisfy the inequality $x^*x + y^*y < n$.

Examples:

Input: n = 5

Output: 6

The pairs are (0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (0, 2)

Input: n = 6

Output: 8

The pairs are (0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (0, 2),
(1, 2), (2, 1)

A **Simple Solution** is to run two loops. The outer loop goes for all possible values of x (from 0 to \sqrt{n}). The inner loop picks all possible values of y for current value of x (picked by outer loop). Following is C++ implementation of simple solution.

```
#include <iostream>
using namespace std;

// This function counts number of pairs (x, y) that satisfy
// the inequality x*x + y*y < n.
int countSolutions(int n)
{
    int res = 0;
    for (int x = 0; x*x < n; x++)
        for (int y = 0; x*x + y*y < n; y++)
            res++;
    return res;
}

// Driver program to test above function
int main()
{
    cout << "Total Number of distinct Non-Negative pairs is "
        << countSolutions(6) << endl;
    return 0;
}
```

[Run on IDE](#)

Output:

Total Number of distinct Non-Negative pairs is 8

An upper bound for time complexity of the above solution is $O(n)$. The outer loop runs \sqrt{n} times. The inner loop runs at most \sqrt{n} times.

Using an **Efficient Solution**, we can find the count in $O(\sqrt{n})$ time. The idea is to first find the count of all y values corresponding the 0 value of x. Let count of distinct y values be yCount. We can find yCount by running a loop and comparing yCount*yCount with n.

After we have initial yCount, we can one by one increase value of x and find the next value of yCount by reducing yCount.

```
// An efficient C program to find different (x, y) pairs that
// satisfy x*x + y*y < n.
#include <iostream>
using namespace std;

// This function counts number of pairs (x, y) that satisfy
// the inequality x*x + y*y < n.
int countSolutions(int n)
{
    int x = 0, yCount, res = 0;

    // Find the count of different y values for x = 0.
    for (yCount = 0; yCount*yCount < n; yCount++) ;

    // One by one increase value of x, and find yCount for
    // current x. If yCount becomes 0, then we have reached
    // maximum possible value of x.
    while (yCount != 0)
    {
        // Add yCount (count of different possible values of y
        // for current x) to result
        res += yCount;

        // Increment x
        x++;

        // Update yCount for current x. Keep reducing yCount while
        // the inequality is not satisfied.
        while (yCount != 0 && (x*x + (yCount-1)*(yCount-1) >= n))
            yCount--;
    }

    return res;
}

// Driver program to test above function
int main()
{
    cout << "Total Number of distinct Non-Negative pairs is "
         << countSolutions(6) << endl;
    return 0;
}
```

Run on IDE

Output:

Total Number of distinct Non-Negative pairs is 8

Time Complexity of the above solution seems more but if we take a closer look, we can see that it is $O(\sqrt{n})$. In every step inside the inner loop, value of yCount is decremented by 1. The value yCount can decrement at most $O(\sqrt{n})$ times as yCount is count y values for $x = 0$. In the outer loop, the value of x is incremented. The value of x can also increment at most $O(\sqrt{n})$ times as the last x is for yCount equals to 1.

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

The advertisement features the Global Sources logo at the top. Below it, the text "Import office calculators from verified suppliers in China". Three images of calculators are shown: two black calculators and one silver calculator. A blue button below the images says "Get prices & compare". At the bottom, a red bar contains the website address "www.globalsources.com".

GATE CS Corner Company Wise Coding Practice

[Mathematical](#) [MathematicalAlgo](#)

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.6 Average Difficulty : **2.6/5.0**
Based on **5** vote(s)

Add to TODO List
 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

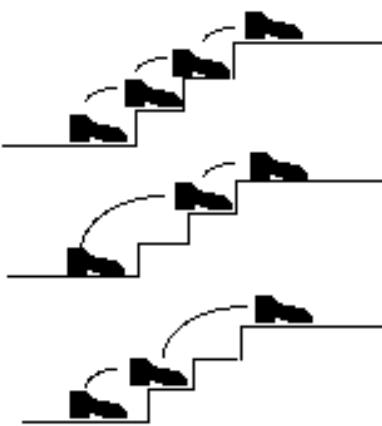
[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Count ways to reach the n'th stair

There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1 stair or 2 stairs at a time. Count the number of ways, the person can reach the top.

Consider the example shown in diagram. The value of n is 3. There are 3 ways to reach the top. The diagram is taken from [Easier Fibonacci puzzles](#)



More Examples:

Input: $n = 1$

Output: 1

There is only one way to climb 1 stair

Input: $n = 2$

Output: 2

There are two ways: (1, 1) and (2)

Input: $n = 4$

Output: 5

(1, 1, 1, 1), (1, 1, 2), (2, 1, 1), (1, 2, 1), (2, 2)

We strongly recommend that you click here and practice it, before moving on to the solution.

We can easily find recursive nature in above problem. The person can reach n 'th stair from either $(n-1)$ 'th stair or from $(n-2)$ 'th stair. Let the total number of ways to reach n 't stair be ' $\text{ways}(n)$ '. The value of ' $\text{ways}(n)$ ' can be written as following.

```
ways(n) = ways(n-1) + ways(n-2)
```

The above expression is actually the expression for **Fibonacci numbers**, but there is one thing to notice, the value of ways(n) is equal to fibonacci(n+1).

```
ways(1) = fib(2) = 1
ways(2) = fib(3) = 2
ways(3) = fib(4) = 3
```

So we can use function for fibonacci numbers to find the value of ways(n). Following is C++ implementation of the above idea.

```
// A C program to count number of ways to reach n'th stair when
// a person can climb 1, 2, ...m stairs at a time.
#include<stdio.h>

// A simple recursive program to find n'th fibonacci number
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}

// Returns number of ways to reach s'th stair
int countWays(int s)
{
    return fib(s + 1);
}

// Driver program to test above functions
int main ()
{
    int s = 4;
    printf("Number of ways = %d", countWays(s));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Java

```
class stairs
{
    // A simple recursive program to find n'th fibonacci number
    static int fib(int n)
    {
        if (n <= 1)
            return n;
        return fib(n-1) + fib(n-2);
    }

    // Returns number of ways to reach s'th stair
    static int countWays(int s)
    {
        return fib(s + 1);
    }

    /* Driver program to test above function */
    public static void main (String args[])
}
```

```

{
    int s = 4;
    System.out.println("Number of ways = "+ countWays(s));
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

Number of ways = 5

The time complexity of the above implementation is exponential (golden ratio raised to power n). It can be optimized to work in O(Logn) time using the previously [discussed Fibonacci function optimizations](#).

Generalization of the above problem

How to count number of ways if the person can climb up to m stairs for a given value m? For example if m is 4, the person can climb 1 stair or 2 stairs or 3 stairs or 4 stairs at a time.

We can write the recurrence as following.

$$\text{ways}(n, m) = \text{ways}(n-1, m) + \text{ways}(n-2, m) + \dots + \text{ways}(n-m, m)$$

Following is C++ implementation of above recurrence.

```

// A C program to count number of ways to reach n'th stair when
// a person can climb either 1 or 2 stairs at a time
#include<stdio.h>

// A recursive function used by countWays
int countWaysUtil(int n, int m)
{
    if (n <= 1)
        return n;
    int res = 0;
    for (int i = 1; i<=m && i<=n; i++)
        res += countWaysUtil(n-i, m);
    return res;
}

// Returns number of ways to reach s'th stair
int countWays(int s, int m)
{
    return countWaysUtil(s+1, m);
}

// Driver program to test above functions
int main ()
{
    int s = 4, m = 2;
    printf("Number of ways = %d", countWays(s, m));
    return 0;
}

```

[Run on IDE](#)

Java

```

class stairs
{
    // A recursive function used by countWays
    static int countWaysUtil(int n, int m)
    {
        if (n <= 1)
            return n;
        int res = 0;
        for (int i = 1; i<=m && i<=n; i++)
            res += countWaysUtil(n-i, m);
        return res;
    }

    // Returns number of ways to reach s'th stair
    static int countWays(int s, int m)
    {
        return countWaysUtil(s+1, m);
    }
}

/* Driver program to test above function */
public static void main (String args[])
{
    int s = 4,m = 2;
    System.out.println("Number of ways = "+ countWays(s,m));
}
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Output:

Number of ways = 5

The time complexity of above solution is exponential. It can be optimized to O(mn) by using dynamic programming. Following is dynamic programming based solution. We build a table res[] in bottom up manner.

```

// A C program to count number of ways to reach n'th stair when
// a person can climb 1, 2, ...m stairs at a time
#include<stdio.h>

// A recursive function used by countWays
int countWaysUtil(int n, int m)
{
    int res[n];
    res[0] = 1; res[1] = 1;
    for (int i=2; i<n; i++)
    {
        res[i] = 0;
        for (int j=1; j<=m && j<=i; j++)
            res[i] += res[i-j];
    }
    return res[n-1];
}

// Returns number of ways to reach s'th stair
int countWays(int s, int m)
{
    return countWaysUtil(s+1, m);
}

// Driver program to test above functions
int main ()
{
    int s = 4, m = 2;

```

```

printf("Nuber of ways = %d", countWays(s, m));
return 0;
}

```

[Run on IDE](#)

Java

```

class stairs
{
    // A recursive function used by countWays
    static int countWaysUtil(int n, int m)
    {
        if (n <= 1)
            return n;
        int res = 0;
        for (int i = 1; i<=m && i<=n; i++)
            res += countWaysUtil(n-i, m);
        return res;
    }
    // Returns number of ways to reach s'th stair
    static int countWays(int s, int m)
    {
        return countWaysUtil(s+1, m);
    }

    /* Driver program to test above function */
    public static void main (String args[])
    {
        int s = 4,m = 2;
        System.out.println("Number of ways = "+ countWays(s,m));
    }
}/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

Number of ways = 5

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



GATE CS Corner Company Wise Coding Practice

Mathematical

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 49 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Replace all '0' with '5' in an input Integer

Given a integer as a input and replace all the '0' with '5' in the integer.

Examples:

```
102 - 152
1020 - 1525
```

Use of array to store all digits is not allowed.

Source: Amazon interview Experience | Set 136 (For SDE-T)

We strongly recommend to minimize the browser and try this yourself first.

The idea is simple, we get the last digit using mod operator '%'. If the digit is 0, we replace it with 5, otherwise keep it as it is. Then we recur for remaining digits.

Following is C implementation of the above idea.

```
// C program to replace all '0' with '5' in an input Integer
#include<stdio.h>

// A recursive function to replace all 0s with 5s in an input number
// It doesn't work if input number itself is 0.
int convert0To5Rec(int num)
{
    // Base case for recursion termination
    if (num == 0)
        return 0;

    // Extract the last digit and change it if needed
    int digit = num % 10;
    if (digit == 0)
        digit = 5;

    // Convert remaining digits and append the last digit
    return convert0To5Rec(num/10) * 10 + digit;
}

// It handles 0 and calls convert0To5Rec() for other numbers
int convert0To5(int num)
{
    if (num == 0)
        return 5;
    else return convert0To5Rec(num);
}

// Driver program to test above function
int main()
{
    int num = 10120;
    printf("%d", convert0To5(num));
    return 0;
}
```

Run on IDE

Output:

15125

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

GATE CS Corner



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

([Login](#) to Rate and Mark)

Average Rating : **0/5.0**

No votes yet.



Average Difficulty : **1.7/5.0**

1.7

Based on **25** vote(s)



Add to TODO List



Mark as DONE

[Load Comments](#)

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Program to add two polynomials

Given two polynomials represented by two arrays, write a function that adds given two polynomials.

Example:

```
Input: A[] = {5, 0, 10, 6}
       B[] = {1, 2, 4}
Output: sum[] = {5, 10, 30, 26, 52, 24}
```

The first input array represents "5 + 0x^1 + 10x^2 + 6x^3"

The second array represents "1 + 2x^1 + 4x^2"

And Output is "6 + 2x^1 + 14x^2 + 6x^3"

We strongly recommend to minimize your browser and try this yourself first.

Addition is simpler than [multiplication of polynomials](#). We initialize result as one of the two polynomials, then we traverse the other polynomial and add all terms to the result.

```
add(A[0..m-1], B[0..n-1])
1) Create a sum array sum[] of size equal to maximum of 'm' and 'n'
2) Copy A[] to sum[].
3) Travers array B[] and do following for every element B[i]
   sum[i] = sum[i] + B[i]
4) Return sum[].
```

The following is C++ implementation of above algorithm.

```
// Simple C++ program to add two polynomials
#include <iostream>
using namespace std;

// A utility function to return maximum of two integers
int max(int m, int n) { return (m > n)? m: n; }

// A[] represents coefficients of first polynomial
// B[] represents coefficients of second polynomial
// m and n are sizes of A[] and B[] respectively
int *add(int A[], int B[], int m, int n)
{
    int size = max(m, n);
    int *sum = new int[size];

    // Initialize the product polynomial
    for (int i = 0; i<m; i++)
        sum[i] = A[i];

    // Take every term of first polynomial
    for (int i=0; i<n; i++)
        sum[i] += B[i];

    return sum;
}

// A utility function to print a polynomial
void printPoly(int poly[], int n)
```

```

{
    for (int i=0; i<n; i++)
    {
        cout << poly[i];
        if (i != 0)
            cout << "x^" << i ;
        if (i != n-1)
            cout << " + ";
    }

// Driver program to test above functions
int main()
{
    // The following array represents polynomial 5 + 10x^2 + 6x^3
    int A[] = {5, 0, 10, 6};

    // The following array represents polynomial 1 + 2x + 4x^2
    int B[] = {1, 2, 4};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);

    cout << "First polynomial is \n";
    printPoly(A, m);
    cout << "\nSecond polynomial is \n";
    printPoly(B, n);

    int *sum = add(A, B, m, n);
    int size = max(m, n);

    cout << "\nSum polynomial is \n";
    printPoly(sum, size);

    return 0;
}

```

[Run on IDE](#)

Output:

```

First polynomial is
5 + 0x^1 + 10x^2 + 6x^3
Second polynomial is
1 + 2x^1 + 4x^2
Sum polynomial is
6 + 2x^1 + 14x^2 + 6x^3

```

Time complexity of the above algorithm and program is $O(m+n)$ where m and n are orders of two given polynomials.

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

GATE CS Corner

Ali Tile BY **LOLA GROUP**

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

([Login](#) to Rate and Mark)

Average Rating : **5/5.0**

Based on 1 vote(s)



Average Difficulty : **1.2/5.0**

1.2

Based on 5 vote(s)

[Add to TODO List](#)

[Mark as DONE](#)

[Load Comments](#)

@geeksforgeeks Some rights reserved

[Contact Us!](#)

[About Us!](#)

[Privacy Policy](#)

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Print first k digits of 1/n where n is a positive integer

Given a positive integer n, print first k digits after point in value of 1/n. Your program should avoid overflow and floating point arithmetic.

Examples:

Input: n = 3, k = 3

Output: 333

Input: n = 50, k = 4

Output: 0200

We strongly recommend to minimize the browser and try this yourself first.

Let us consider an example n = 7, k = 3. The first digit of 1/7 is '1', it can be obtained by doing integer value of 10/7. Remainder of 10/7 is 3. Next digit is 4 which can be obtained by taking integer value of 30/7. Remainder of 30/7 is 2. Next digits is 2 which can be obtained by taking integer value of 20/7

```
#include <iostream>
using namespace std;

// Function to print first k digits after dot in value
// of 1/n. n is assumed to be a positive integer.
void print(int n, int k)
{
    int rem = 1; // Initialize remainder

    // Run a loop k times to print k digits
    for (int i = 0; i < k; i++)
    {
        // The next digit can always be obtained as
        // doing (10*rem)/10
        cout << (10 * rem) / n;

        // Update remainder
        rem = (10*rem) % n;
    }
}

// Driver program to test above function
int main()
{
    int n = 7, k = 3;
    print(n, k);
    cout << endl;

    n = 21, k = 4;
    print(n, k);

    return 0;
}
```

Run on IDE

Output:

```
142  
0476
```

Reference:

Algorithms And Programming: Problems And Solutions by Alexander Shen

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

GATE CS Corner



Download

Free Download

unzipper.com



See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

([Login](#) to Rate and Mark)

Average Rating : 0/5.0

No votes yet.



Average Difficulty : 2.4/5.0

2.4

Based on 5 vote(s)



Add to TODO List



Mark as DONE

Load Comments

GeeksQuiz

Computer Science Quizzes for Geeks !

[GATE CS](#) [Coding Practice](#) [Placements](#) [GeeksforGeeks](#)

Given a number as a string, find the number of contiguous subsequences which recursively add up to 9

Given a number as a string, write a function to find the number of substrings (or contiguous subsequences) of the given string which recursively add up to 9.

For example digits of 729 recursively add to 9,

$$7 + 2 + 9 = 18$$

Recur for 18

$$1 + 8 = 9$$

Examples:

```
Input: 4189
Output: 3
There are three substrings which recursively add to 9.
The substrings are 18, 9 and 189.
```

```
Input: 999
Output: 6
There are 6 substrings which recursively add to 9.
9, 99, 999, 9, 99, 9
```

All digits of a number recursively add up to 9, if only if the number is multiple of 9. We basically need to check for $s \% 9$ for all substrings s . One trick used in below program is to do modular arithmetic to avoid overflow for big strings.

Following is a simple implementation based on this approach. The implementation assumes that there are no leading 0's in input number.

```
// C++ program to count substrings with recursive sum equal to 9
#include <iostream>
#include <cstring>
using namespace std;

int count9s(char number[])
{
    int count = 0; // To store result
    int n = strlen(number);

    // Consider every character as beginning of substring
    for (int i = 0; i < n; i++)
    {
        int sum = number[i] - '0'; //sum of digits in current substring

        if (number[i] == '9') count++;

        // One by one choose every character as an ending character
    }
}
```

```

for (int j = i+1; j < n; j++)
{
    // Add current digit to sum, if sum becomes multiple of 5
    // then increment count. Let us do modular arithmetic to
    // avoid overflow for big strings
    sum = (sum + number[j] - '0')%9;

    if (sum == 0)
        count++;
}
return count;
}

// driver program to test above function
int main()
{
    cout << count9s("4189") << endl;
    cout << count9s("1809");
    return 0;
}

```

[Run on IDE](#)

Output:

```

3
5

```

Time complexity of the above program is $O(n^2)$. Please let me know if there is a better solution.

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

GATE CS Corner



China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

See [Placement Course](#) for placement preparation, [GATE Corner](#) for GATE CS Preparation and [Quiz Corner](#) for all Quizzes on GeeksQuiz.

Category: Algorithms

(Login to Rate and Mark)

Average Rating : 0/5.0

No votes yet.

Average Difficulty : 2/5.0

2

Based on 3 vote(s)



Add to TODO List



Mark as DONE

Load Comments

@geeksforgeeks Some rights reserved

Contact Us!

About Us!

Privacy Policy

HANDBOOK OF ALGORITHMS

Section
Geometry Algorithms

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Closest Pair of Points | O(nlogn) Implementation

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

We have discussed a [divide and conquer solution](#) for this problem. The time complexity of the implementation provided in the previous post is $O(n (\log n)^2)$. In this post, we discuss an implementation with time complexity as $O(n \log n)$.

Following is a recap of the algorithm discussed in the previous post.

- 1) We sort all points according to x coordinates.
- 2) Divide all points in two halves.
- 3) Recursively find the smallest distances in both subarrays.
- 4) Take the minimum of two smallest distances. Let the minimum be d.
- 5) Create an array strip[] that stores all points which are at most d distance away from the middle line dividing the two sets.
- 6) Find the smallest distance in strip[].
- 7) Return the minimum of d and the smallest distance calculated in above step 6.

The great thing about the above approach is, if the array strip[] is sorted according to y coordinate, then we can find the smallest distance in strip[] in $O(n)$ time. In the implementation discussed in previous post, strip[] was explicitly sorted in every recursive call that made the time complexity $O(n (\log n)^2)$, assuming that the sorting step takes $O(n \log n)$ time.

In this post, we discuss an implementation where the time complexity is $O(n \log n)$. The idea is to presort all points according to y coordinates. Let the sorted array be Py[]. When we make recursive calls, we need to divide points of Py[] also according to the vertical line. We can do that by simply processing every point and comparing its x coordinate with x coordinate of middle line.

Following is C++ implementation of $O(n \log n)$ approach.

```
// A divide and conquer program in C++ to find the smallest distance from a
// given set of points.
```

```

#include <iostream>
#include <float.h>
#include <stdlib.h>
#include <math.h>
using namespace std;

// A structure to represent a Point in 2D plane
struct Point
{
    int x, y;
};

/* Following two functions are needed for library function qsort().
   Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */

// Needed to sort array of points according to X coordinate
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

// A utility function to find the distance between two points
float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y) );
}

// A Brute Force method to return the smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}

// A utility function to find minimum of two float values
float min(float x, float y)
{
    return (x < y)? x : y;
}

// A utility function to find the distance between the closest points of
// strip of given size. All points in strip[] are sorted according to
// y coordinate. They all have an upper bound on minimum distance as d.
// Note that this method seems to be a O(n^2) method, but it's a O(n)
// method as the inner loop runs at most 6 times
float stripClosest(Point strip[], int size, float d)
{
    float min = d; // Initialize the minimum distance as d

    // Pick all points one by one and try the next points till the difference
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);
}

```

```

    return min;
}

// A recursive function to find the smallest distance. The array Px contains
// all points sorted according to x coordinates and Py contains all points
// sorted according to y coordinates
float closestUtil(Point Px[], Point Py[], int n)
{
    // If there are 2 or 3 points, then use brute force
    if (n <= 3)
        return bruteForce(Px, n);

    // Find the middle point
    int mid = n/2;
    Point midPoint = Px[mid];

    // Divide points in y sorted array around the vertical line.
    // Assumption: All x coordinates are distinct.
    Point Pyl[mid+1]; // y sorted points on left of vertical line
    Point Pyr[n-mid-1]; // y sorted points on right of vertical line
    int li = 0, ri = 0; // indexes of left and right subarrays
    for (int i = 0; i < n; i++)
    {
        if (Py[i].x <= midPoint.x)
            Pyl[li++] = Py[i];
        else
            Pyr[ri++] = Py[i];
    }

    // Consider the vertical line passing through the middle point
    // calculate the smallest distance dl on left of middle point and
    // dr on right side
    float dl = closestUtil(Px, Pyl, mid);
    float dr = closestUtil(Px + mid, Pyr, n-mid);

    // Find the smaller of two distances
    float d = min(dl, dr);

    // Build an array strip[] that contains points close (closer than d)
    // to the line passing through the middle point
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(Py[i].x - midPoint.x) < d)
            strip[j] = Py[i], j++;

    // Find the closest points in strip. Return the minimum of d and closest
    // distance is strip[]
    return min(d, stripClosest(strip, j, d));
}

// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n)
{
    Point Px[n];
    Point Py[n];
    for (int i = 0; i < n; i++)
    {
        Px[i] = P[i];
        Py[i] = P[i];
    }

    qsort(Px, n, sizeof(Point), compareX);
    qsort(Py, n, sizeof(Point), compareY);

    // Use recursive function closestUtil() to find the smallest distance
    return closestUtil(Px, Py, n);
}

// Driver program to test above functions
int main()
{

```

```
Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
int n = sizeof(P) / sizeof(P[0]);
cout << "The smallest distance is " << closest(P, n);
return 0;
}
```

[Run on IDE](#)

Output:

```
The smallest distance is 1.41421
```

Time Complexity: Let Time complexity of above algorithm be $T(n)$. Let us assume that we use a $O(n \log n)$ sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in $O(n)$ time. Also, it takes $O(n)$ time to divide the Py array around the mid vertical line. Finally finds the closest points in strip in $O(n)$ time. So $T(n)$ can be expressed as follows

$$T(n) = 2T(n/2) + O(n) + O(n) + O(n)$$

$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = T(n \log n)$$

References:

<http://www.cs.umd.edu/class/fall2013/cmsc451/Lects/lect10.pdf>

<http://www.youtube.com/watch?v=vS4Zn1a9KUc>

<http://www.youtube.com/watch?v=T3T7T8Ym20M>

http://en.wikipedia.org/wiki/Closest_pair_of_points_problem

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Divide and Conquer Geometric Divide and Conquer geometric algorithms

Related Posts:

- Minimum difference between adjacent elements of array which contain elements from each row of a matrix
- Find bitonic point in given bitonic sequence
- Find the only repeating element in a sorted array of size n
- Floor in a Sorted Array
- Find cubic root of a number
- Find frequency of each element in a limited range array in less than O(n) time
- Longest Common Prefix | Set 3 (Divide and Conquer)
- Square root of an integer

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 13 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

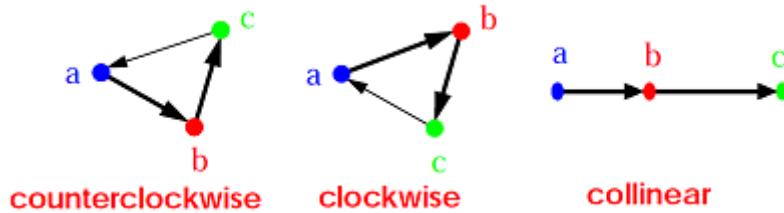
How to check if two given line segments intersect?

Given two line segments (p_1, q_1) and (p_2, q_2) , find if the given line segments intersect with each other.

Before we discuss solution, let us define notion of **orientation**. Orientation of an ordered triplet of points in the plane can be

- counterclockwise
- clockwise
- colinear

The following diagram shows different possible orientations of (a, b, c)



We strongly recommend that you click here and practice it, before moving on to the solution.

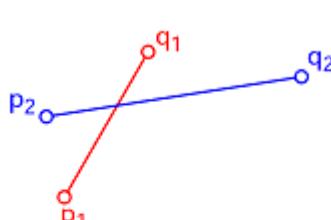
How is Orientation useful here?

Two segments (p_1, q_1) and (p_2, q_2) intersect if and only if one of the following two conditions is verified

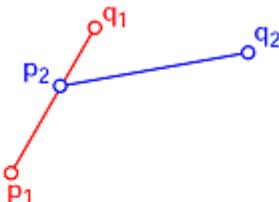
1. General Case:

- (p_1, q_1, p_2) and (p_1, q_1, q_2) have different orientations and
- (p_2, q_2, p_1) and (p_2, q_2, q_1) have different orientations.

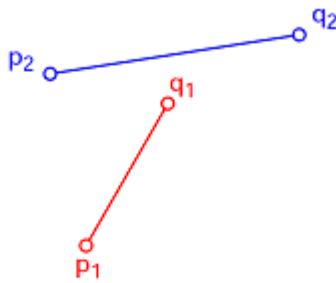
Examples:



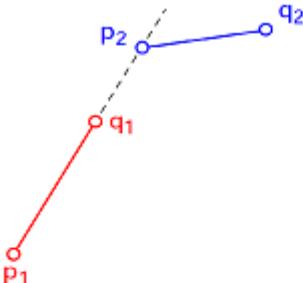
Example 1: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) are also different



Example 2: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) are also different



Example 3: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) are same

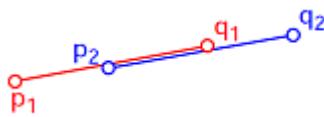


Example 4: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) are same

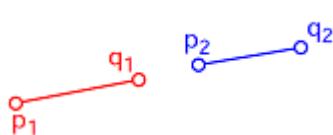
2. Special Case

- (p_1, q_1, p_2) , (p_1, q_1, q_2) , (p_2, q_2, p_1) , and (p_2, q_2, q_1) are all collinear and
- the x-projections of (p_1, q_1) and (p_2, q_2) intersect
- the y-projections of (p_1, q_1) and (p_2, q_2) intersect

Examples:



Example 1: All points are collinear. The x-projections of (p_1, q_1) and (p_2, q_2) intersect. The y-projections of (p_1, q_1) and (p_2, q_2) intersect



Example 2: All points are collinear. The x-projections of (p_1, q_1) and (p_2, q_2) do not intersect. The y-projections of (p_1, q_1) and (p_2, q_2) intersect

Following is C++ implementation based on above idea.

```
// A C++ program to check if two given line segments intersect
#include <iostream>
using namespace std;

struct Point
{
    int x;
    int y;
};

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;

    return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    // See http://www.geeksforgeeks.org/orientation-3-ordered-points/
    // for details of below formula.
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    else if (val < 0) return 1; // Clockwise
    else return 2; // Counterclockwise
}
```

```

    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;

    // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false; // Doesn't fall in any of the above cases
}

// Driver program to test above functions
int main()
{
    struct Point p1 = {1, 1}, q1 = {10, 1};
    struct Point p2 = {1, 2}, q2 = {10, 2};

    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {10, 0}, q1 = {0, 10};
    p2 = {0, 0}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {-5, -5}, q1 = {0, 0};
    p2 = {1, 1}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    return 0;
}

```

[Run on IDE](#)

Output:

No
Yes
No

Asked in: Adobe, Snapdeal, Zomato

Sources:

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Geometric](#) [geometric algorithms](#) [MathematicalAlgo](#)

Related Posts:

- Find perimeter of shapes formed with 1s in binary matrix
- Line Clipping | Set 1 (Cohen–Sutherland Algorithm)
- Find all sides of a right angled triangle from given hypotenuse and area
- Circle and Lattice Points
- n'th Pentagonal Number
- Maximum height when coins are arranged in a triangle
- Classify a triangle
- Optimum location of point to minimize total distance

(Login to Rate and Mark)

3.7

Average Difficulty : 3.7/5.0
Based on 39 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

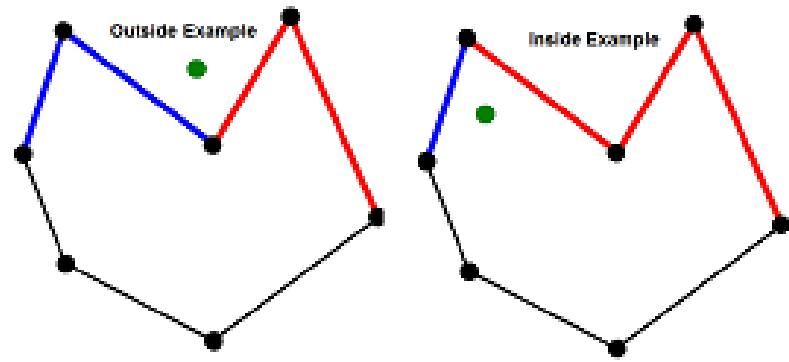
Google™ Custom Search



[Login/Register](#)

How to check if a given point lies inside or outside a polygon?

Given a polygon and a point 'p', find if 'p' lies inside the polygon or not. The points lying on the border are considered inside.

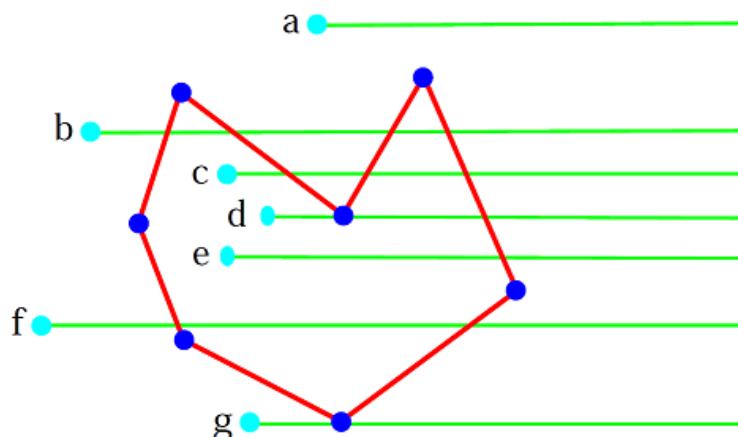


We strongly recommend to see the following post first.

[How to check if two given line segments intersect?](#)

Following is a simple idea to check whether a point is inside or outside.

- 1) Draw a horizontal line to the right of each point and extend it to infinity
- 1) Count the number of times the line intersects with polygon edges.
- 2) A point is inside the polygon if either count of intersections is odd or point lies on an edge of polygon. If none of the conditions is true, then point lies outside.



How to handle point 'g' in the above figure?

Note that we should return true if the point lies on the line or same as one of the vertices of the given polygon. To handle this, after checking if the line from 'p' to extreme intersects, we check whether 'p' is colinear with vertices of current line of polygon. If it is colinear, then we check if the point 'p' lies on current side of polygon, if it lies, we return true, else false.

Following is C++ implementation of the above idea.

```
// A C++ program to check if a given point lies inside a given polygon
// Refer http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/
// for explanation of functions onSegment(), orientation() and doIntersect()
#include <iostream>
using namespace std;

// Define Infinite (Using INT_MAX caused overflow problems)
#define INF 1000

struct Point
{
    int x;
    int y;
};

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
}
```

```

// p2, q2 and q1 are colinear and q1 lies on segment p2q2
if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false; // Doesn't fall in any of the above cases
}

// Returns true if the point p lies inside the polygon[] with n vertices
bool isInside(Point polygon[], int n, Point p)
{
    // There must be at least 3 vertices in polygon[]
    if (n < 3) return false;

    // Create a point for line segment from p to infinite
    Point extreme = {INF, p.y};

    // Count intersections of the above line with sides of polygon
    int count = 0, i = 0;
    do
    {
        int next = (i+1)%n;

        // Check if the line segment from 'p' to 'extreme' intersects
        // with the line segment from 'polygon[i]' to 'polygon[next]'
        if (doIntersect(polygon[i], polygon[next], p, extreme))
        {
            // If the point 'p' is colinear with line segment 'i-next',
            // then check if it lies on segment. If it lies, return true,
            // otherwise false
            if (orientation(polygon[i], p, polygon[next]) == 0)
                return onSegment(polygon[i], p, polygon[next]);

            count++;
        }
        i = next;
    } while (i != 0);

    // Return true if count is odd, false otherwise
    return count&1; // Same as (count%2 == 1)
}

// Driver program to test above functions
int main()
{
    Point polygon1[] = {{0, 0}, {10, 0}, {10, 10}, {0, 10}};
    int n = sizeof(polygon1)/sizeof(polygon1[0]);
    Point p = {20, 20};
    isInside(polygon1, n, p)? cout << "Yes \n": cout << "No \n";

    p = {5, 5};
    isInside(polygon1, n, p)? cout << "Yes \n": cout << "No \n";

    Point polygon2[] = {{0, 0}, {5, 5}, {5, 0}};
    p = {3, 3};
    n = sizeof(polygon2)/sizeof(polygon2[0]);
    isInside(polygon2, n, p)? cout << "Yes \n": cout << "No \n";

    p = {5, 1};
    isInside(polygon2, n, p)? cout << "Yes \n": cout << "No \n";

    p = {8, 1};
    isInside(polygon2, n, p)? cout << "Yes \n": cout << "No \n";

    Point polygon3[] = {{0, 0}, {10, 0}, {10, 10}, {0, 10}};
    p = {-1,10};
    n = sizeof(polygon3)/sizeof(polygon3[0]);
    isInside(polygon3, n, p)? cout << "Yes \n": cout << "No \n";

    return 0;
}

```

Run on IDE

Output:

No
Yes
Yes
Yes
No
No

Time Complexity: O(n) where n is the number of vertices in the given polygon.

Source:

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Geometric geometric algorithms MathematicalAlgo

Related Posts:

- Find perimeter of shapes formed with 1s in binary matrix
- Line Clipping | Set 1 (Cohen–Sutherland Algorithm)
- Find all sides of a right angled triangle from given hypotenuse and area
- Circle and Lattice Points
- n'th Pentagonal Number
- Maximum height when coins are arranged in a triangle
- Classify a triangle
- Optimum location of point to minimize total distance

[\(Login to Rate and Mark\)](#)**3.7**Average Difficulty : **3.7/5.0**
Based on **13** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

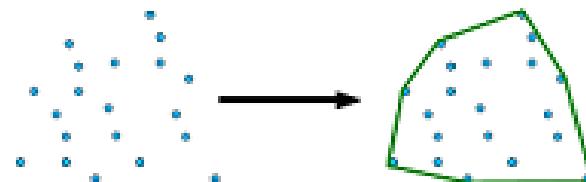
Google™ Custom Search



[Login/Register](#)

Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping)

Given a set of points in the plane, the convex hull of the set is the smallest convex polygon that contains all the points of it.



We strongly recommend to see the following post first.

[How to check if two given line segments intersect?](#)

The idea of Jarvis's Algorithm is simple, we start from the leftmost point (or point with minimum x coordinate value) and we keep wrapping points in counterclockwise direction. The big question is, given a point p as current point, how to find the next point in output? The idea is to use [orientation\(\)](#) here. Next point is selected as the point that beats all other points at counterclockwise orientation, i.e., next point is q if for any other point r, we have "orientation(p, r, q) = counterclockwise". Following is the detailed algorithm.

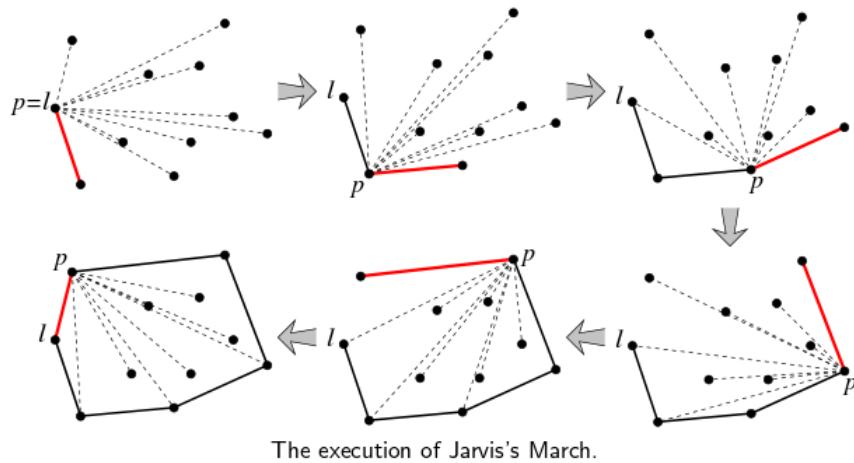
1) Initialize p as leftmost point.

2) Do following while we don't come back to the first (or leftmost) point.

.....**a)** The next point q is the point such that the triplet (p, q, r) is counterclockwise for any other point r.

.....**b)** next[p] = q (Store q as next of p in the output convex hull).

.....**c)** p = q (Set p as q for next iteration).



Below is C++ implementation of above algorithm.

```
// A C++ program to find convex hull of a set of points. Refer
// http://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()
```

```

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

struct Point
{
    int x, y;
};

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // There must be at least 3 points
    if (n < 3) return;

    // Initialize Result
    vector<Point> hull;

    // Find the leftmost point
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Start from leftmost point, keep moving counterclockwise
    // until reach the start point again. This loop runs O(h)
    // times where h is number of points in result or output.
    int p = l, q;
    do
    {
        // Add current point to result
        hull.push_back(points[p]);

        // Search for a point 'q' such that orientation(p, x,
        // q) is counterclockwise for all points 'x'. The idea
        // is to keep track of last visited most counterclock-
        // wise point in q. If any point 'i' is more counterclock-
        // wise than q, then update q.
        q = (p+1)%n;
        for (int i = 0; i < n; i++)
        {
            // If i is more counterclockwise than current q, then
            // update q
            if (orientation(points[p], points[i], points[q]) == 2)
                q = i;
        }

        // Now q is the most counterclockwise with respect to p
        // Set p as q for next iteration, so that q is added to
        // result 'hull'
        p = q;
    } while (p != l); // While we don't come to first point

    // Print Result
    for (int i = 0; i < hull.size(); i++)
        cout << "(" << hull[i].x << ", "
             << hull[i].y << ")\n";
}

// Driver program to test above functions

```

```

int main()
{
    Point points[] = {{0, 3}, {2, 2}, {1, 1}, {2, 1},
                      {3, 0}, {0, 0}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}

```

[Run on IDE](#)

Output: The output is points of the convex hull.

```
(0, 3)
(0, 0)
(3, 0)
(3, 3)
```

Time Complexity: For every point on the hull we examine all the other points to determine the next point. Time complexity is $?m * n$ where n is number of input points and m is number of output or hull points ($m \leq n$). In worst case, time complexity is $O(n^2)$. The worst case occurs when all the points are on the hull ($m = n$)

Set 2- Convex Hull (Graham Scan)

Sources:

<http://www.cs.uiuc.edu/~jeffe/teaching/373/notes/x05-convexhull.pdf>

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Geometric | geometric algorithms | MathematicalAlgo

Related Posts:

- Find perimeter of shapes formed with 1s in binary matrix
- Line Clipping | Set 1 (Cohen–Sutherland Algorithm)
- Find all sides of a right angled triangle from given hypotenuse and area
- Circle and Lattice Points
- n'th Pentagonal Number
- Maximum height when coins are arranged in a triangle
- Classify a triangle
- Optimum location of point to minimize total distance

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 9 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

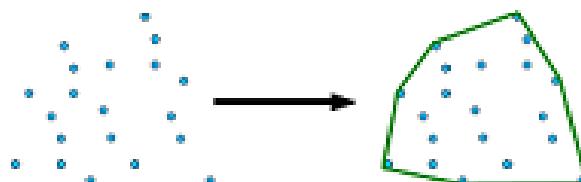
Google™ Custom Search



[Login/Register](#)

Convex Hull | Set 2 (Graham Scan)

Given a set of points in the plane. the convex hull of the set is the smallest convex polygon that contains all the points of it.



We strongly recommend to see the following post first.

[How to check if two given line segments intersect?](#)

We have discussed [Jarvis's Algorithm](#) for Convex Hull. Worst case time complexity of Jarvis's Algorithm is $O(n^2)$.

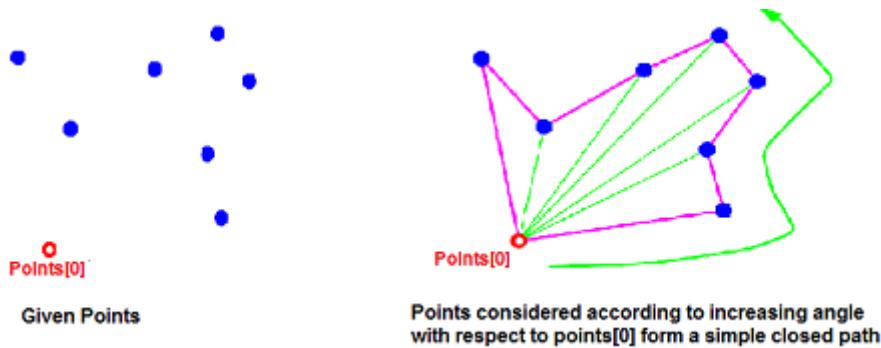
Using Graham's scan algorithm, we can find Convex Hull in $O(n \log n)$ time. Following is Graham's algorithm

Let $\text{points}[0..n-1]$ be the input array.

- 1) Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered. Let the bottom-most point be P_0 . Put P_0 at first position in output hull.
- 2) Consider the remaining $n-1$ points and sort them by polar angle in counterclockwise order around $\text{points}[0]$. If polar angle of two points is same, then put the nearest point first.
- 3) After sorting, check if two or more points have same angle. If two more points have same angle, then remove all same angle points except the point farthest from P_0 . Let the size of new array be m .
- 4) If m is less than 3, return (Convex Hull not possible)
- 5) Create an empty stack 'S' and push $\text{points}[0]$, $\text{points}[1]$ and $\text{points}[2]$ to S.
- 6) Process remaining $m-3$ points one by one. Do following for every point ' $\text{points}[i]$ '
 4.1) Keep removing points from stack while [orientation](#) of following 3 points is not counterclockwise (or they don't make a left turn).
 - a) Point next to top in stack
 - b) Point at the top of stack
 - c) $\text{points}[i]$
 4.2) Push $\text{points}[i]$ to S
- 5) Print contents of S

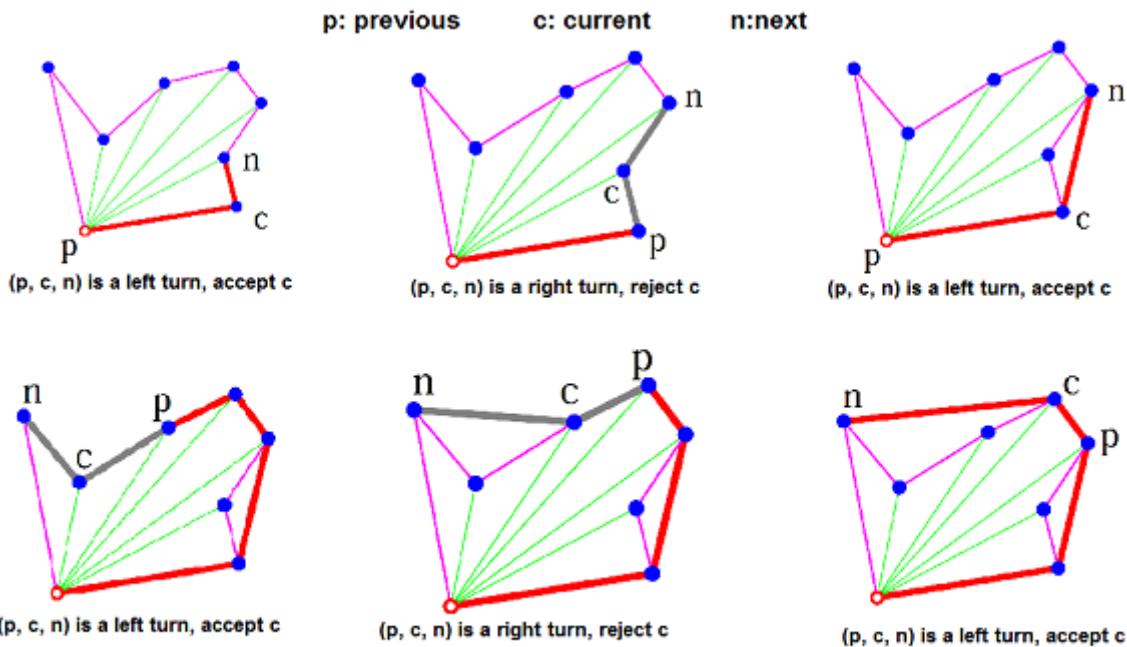
The above algorithm can be divided in two phases.

Phase 1 (Sort points): We first find the bottom-most point. The idea is to pre-process points by sorting them with respect to the bottom-most point. Once the points are sorted, they form a simple closed path (See following diagram).



What should be the sorting criteria? computation of actual angles would be inefficient since trigonometric functions are not simple to evaluate. The idea is to use the orientation to compare angles without actually computing them (See the compare() function below)

Phase 2 (Accept or Reject Points): Once we have the closed path, the next step is to traverse the path and remove concave points on this path. How to decide which point to remove and which to keep? Again, orientation helps here. The first two points in sorted array are always part of Convex Hull. For remaining points, we keep track of recent three points, and find the angle formed by them. Let the three points be prev(p), curr(c) and next(n). If orientation of these points (considering them in same order) is not counterclockwise, we discard c, otherwise we keep it. Following diagram shows step by step process of this phase (Source of these diagrams is [Ref 2](#)).



In the above algorithm and below code, a stack of points is used to store convex hull points. With reference to the code, p is next-to-top in stack, c is top of stack and n is points[i].

Following is C++ implementation of the above algorithm.

```
// A C++ program to find convex hull of a set of points. Refer
// http://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;
```

```

struct Point
{
    int x, y;
};

// A globle point needed for sorting points with reference
// to the first point Used in compare function of qsort()
Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance
// between p1 and p2
int distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
           (p1.y - p2.y)*(p1.y - p2.y);
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// A function used by library function qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2)
{
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

    return (o == 2)? -1: 1;
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // Find the bottommost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++)
    {
        int y = points[i].y;

        // Pick the bottom-most or chose the left
}

```

```

// most point in case of tie
if ((y < ymin) || (ymin == y &&
    points[i].x < points[min].x))
    ymin = points[i].y, min = i;
}

// Place the bottom-most point at first position
swap(points[0], points[min]);

// Sort n-1 points with respect to the first point.
// A point p1 comes before p2 in sorted output if p2
// has larger polar angle (in counterclockwise
// direction) than p1
p0 = points[0];
qsort(&points[1], n-1, sizeof(Point), compare);

// If two or more points make same angle with p0,
// Remove all but the one that is farthest from p0
// Remember that, in above sorting, our criteria was
// to keep the farthest point at the end when more than
// one points have same angle.
int m = 1; // Initialize size of modified array
for (int i=1; i<n; i++)
{
    // Keep removing i while angle of i and i+1 is same
    // with respect to p0
    while (i < n-1 && orientation(p0, points[i],
                                    points[i+1]) == 0)
        i++;

    points[m] = points[i];
    m++; // Update size of modified array
}

// If modified array of points has less than 3 points,
// convex hull is not possible
if (m < 3) return;

// Create an empty stack and push first three points
// to it.
stack<Point> S;
S.push(points[0]);
S.push(points[1]);
S.push(points[2]);

// Process remaining n-3 points
for (int i = 3; i < m; i++)
{
    // Keep removing top while the angle formed by
    // points next-to-top, top, and points[i] makes
    // a non-left turn
    while (orientation(nextToTop(S), S.top(), points[i]) != 2)
        S.pop();
    S.push(points[i]);
}

// Now stack has the output points, print contents of stack
while (!S.empty())
{
    Point p = S.top();
    cout << "(" << p.x << ", " << p.y << ")" << endl;
    S.pop();
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                      {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}

```

}

[Run on IDE](#)**Output:**

```
(0, 3)
(4, 4)
(3, 1)
(0, 0)
```

Time Complexity: Let n be the number of input points. The algorithm takes $O(n\log n)$ time if we use a $O(n\log n)$ sorting algorithm.

The first step (finding the bottom-most point) takes $O(n)$ time. The second step (sorting points) takes $O(n\log n)$ time. Third step takes $O(n)$ time. In third step, every element is pushed and popped at most one time. So the sixth step to process points one by one takes $O(n)$ time, assuming that the stack operations take $O(1)$ time. Overall complexity is $O(n) + O(n\log n) + O(n) + O(n)$ which is $O(n\log n)$

References:

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
<http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Geometric](#) [geometric algorithms](#) [MathematicalAlgo](#)

Related Posts:

- Find perimeter of shapes formed with 1s in binary matrix
- Line Clipping | Set 1 (Cohen–Sutherland Algorithm)
- Find all sides of a right angled triangle from given hypotenuse and area

- Circle and Lattice Points
- n'th Pentagonal Number
- Maximum height when coins are arranged in a triangle
- Classify a triangle
- Optimum location of point to minimize total distance

(Login to Rate and Mark)

2.6

Average Difficulty : **2.6/5.0**
Based on **6** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Given n line segments, find if any two segments intersect

We have discussed the problem to detect if [two given line segments intersect or not](#). In this post, we extend the problem. Here we are given n line segments and we need to find out if any two line segments intersect or not.

Naive Algorithm A naive solution to solve this problem is to check every pair of lines and check if the pair intersects or not. [We can check two line segments in O\(1\) time](#). Therefore, this approach takes $O(n^2)$.

Sweep Line Algorithm: We can solve this problem in $O(n \log n)$ time using Sweep Line Algorithm. The algorithm first sorts the end points along the x axis from left to right, then it passes a vertical line through all points from left to right and checks for intersections. Following are detailed steps.

1) Let there be n given lines. There must be $2n$ end points to represent the n lines. Sort all points according to x coordinates. While sorting maintain a flag to indicate whether this point is left point of its line or right point.

2) Start from the leftmost point. Do following for every point

.....**a)** If the current point is a left point of its line segment, check for intersection of its line segment with the segments just above and below it. And add its line to *active* line segments (line segments for which left end point is seen, but right end point is not seen yet). Note that we consider only those neighbors which are still active.

....**b)** If the current point is a right point, remove its line segment from active list and check whether its two active neighbors (points just above and below) intersect with each other.

The step 2 is like passing a vertical line from all points starting from the leftmost point to the rightmost point. That is why this algorithm is called Sweep Line Algorithm. The Sweep Line technique is useful in many other geometric algorithms like [calculating the 2D Voronoi diagram](#)

What data structures should be used for efficient implementation?

In step 2, we need to store all active line segments. We need to do following operations efficiently:

- a) Insert a new line segment
- b) Delete a line segment
- c) Find predecessor and successor according to y coordinate values

The obvious choice for above operations is Self-Balancing Binary Search Tree like AVL Tree, Red Black Tree. With a Self-Balancing BST, we can do all of the above operations in $O(\log n)$ time.

Also, in step 1, instead of sorting, we can use min heap data structure. Building a min heap takes $O(n)$ time and every extract min operation takes $O(\log n)$ time (See [this](#)).

PseudoCode:

The following pseudocode doesn't use heap. It simply sort the array.

```
sweepLineIntersection(Point[0..2n-1]):
1. Sort Points[] from left to right (according to x coordinate)

2. Create an empty Self-Balancing BST T. It will contain all active line
Segments ordered by y coordinate.

// Process all 2n points
3. for i = 0 to 2n-1

    // If this point is left end of its line
    if (Points[i].isLeft)
        T.insert(Points[i].line()) // Insert into the tree

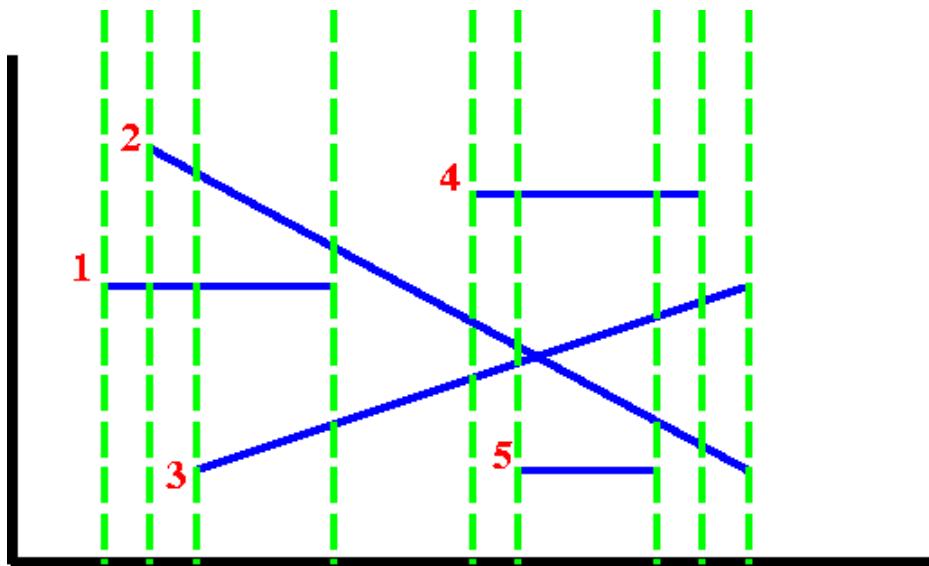
    // Check if this points intersects with its predecessor and successor
    if ( doIntersect(Points[i].line(), T.pred(Points[i].line())) )
        return true
    if ( doIntersect(Points[i].line(), T.succ(Points[i].line())) )
        return true

    else // If it's a right end of its line
        // Check if its predecessor and successor intersect with each other
        if ( doIntersect(T.pred(Points[i].line()), T.succ(Points[i].line())))
            return true
        T.delete(Points[i].line()) // Delete from tree

4. return False
```

Example:

Let us consider the following example taken from [here](#). There are 5 line segments 1, 2, 3, 4 and 5. The dotted green lines show sweep lines.



sweep lines

Following are steps followed by the algorithm. All points from left to right are processed one by one. We maintain a self-balancing binary search tree.

Left end point of line segment 1 is processed: 1 is inserted into the Tree. The tree contains 1. No intersection.

Left end point of line segment 2 is processed: Intersection of 1 and 2 is checked. 2 is inserted into the Tree. No intersection. The tree contains 1, 2.

Left end point of line segment 3 is processed: Intersection of 3 with 1 is checked. No intersection. 3 is inserted into the Tree. The tree contains 2, 1, 3.

Right end point of line segment 1 is processed: 1 is deleted from the Tree. Intersection of 2 and 3 is checked. Intersection of 2 and 3 is reported. The tree contains 2, 3. Note that the above pseudocode returns at this point. We can continue from here to report all intersection points.

Left end point of line segment 4 is processed: Intersections of line 4 with lines 2 and 3 are checked. No intersection. 4 is inserted into the Tree. The tree contains 2, 4, 3.

Left end point of line segment 5 is processed: Intersection of 5 with 3 is checked. No intersection. 4 is inserted into the Tree. The tree contains 2, 4, 3, 5.

Right end point of line segment 5 is processed: 5 is deleted from the Tree. The tree contains 2, 4, 3.

Right end point of line segment 4 is processed: 4 is deleted from the Tree. The tree contains 2, 4, 3. Intersection of 2 with 3 is checked. Intersection of 2 with 3 is reported. The tree contains 2, 3. Note that the intersection of 2 and 3 is reported again. We can add some logic to check for duplicates.

Right end point of line segment 2 and 3 are processed: Both are deleted from tree and tree becomes empty.

Time Complexity: The first step is sorting which takes $O(n\log n)$ time. The second step process $2n$ points and for processing every point, it takes $O(\log n)$ time. Therefore, overall time complexity is $O(n\log n)$

References:

<http://www.cs.uiuc.edu/~jeffe/teaching/373/notes/x06-sweepline.pdf>

<http://courses.csail.mit.edu/6.006/spring11/lectures/lec24.pdf>

<http://www.youtube.com/watch?v=dePDHVovJIE> 

<http://www.eecs.wsu.edu/~cook/aa/lectures/l25/node10.html>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Geometric geometric algorithms

Related Posts:

- Find perimeter of shapes formed with 1s in binary matrix
- Line Clipping | Set 1 (Cohen–Sutherland Algorithm)
- Find all sides of a right angled triangle from given hypotenuse and area
- Circle and Lattice Points
- n'th Pentagonal Number
- Maximum height when coins are arranged in a triangle
- Classify a triangle
- Optimum location of point to minimize total distance

(Login to Rate and Mark)

4.8

Average Difficulty : **4.8/5.0**
Based on 7 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

HANDBOOK OF ALGORITHMS

Section
Dynamic Programming

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 1 (Overlapping Subproblems Property)

Dynamic Programming is an algorithmic paradigm that solves a given complex problem by breaking it into subproblems and stores the results of subproblems to avoid computing the same results again. Following are the two main properties of a problem that suggest that the given problem can be solved using Dynamic programming.

In this post, we will discuss first property (Overlapping Subproblems) in detail. The second property of Dynamic programming is discussed in next post i.e. [Set 2](#).

- 1) Overlapping Subproblems
- 2) Optimal Substructure

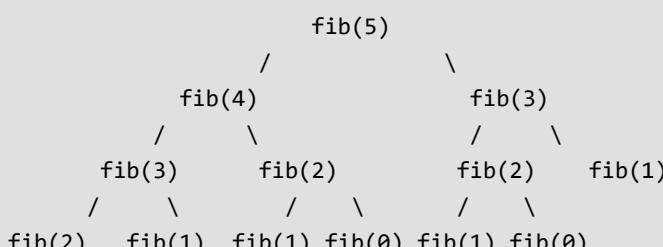
1) Overlapping Subproblems:

Like Divide and Conquer, Dynamic Programming combines solutions to sub-problems. Dynamic Programming is mainly used when solutions of same subproblems are needed again and again. In dynamic programming, computed solutions to subproblems are stored in a table so that these don't have to recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again. For example, [Binary Search](#) doesn't have common subproblems. If we take example of following recursive program for Fibonacci Numbers, there are many subproblems which are solved again and again.

```
/* simple recursive program for Fibonacci numbers */
int fib(int n)
{
    if ( n <= 1 )
        return n;
    return fib(n-1) + fib(n-2);
}
```

[Run on IDE](#)

Recursion tree for execution of `fib(5)`



```
    /   \
fib(1) fib(0)
```

We can see that the function $f(3)$ is being called 2 times. If we would have stored the value of $f(3)$, then instead of computing it again, we could have reused the old stored value. There are following two different ways to store the values so that these values can be reused:

- a) Memoization (Top Down)
- b) Tabulation (Bottom Up)

a) Memoization (Top Down): The memoized program for a problem is similar to the recursive version with a small modification that it looks into a lookup table before computing solutions. We initialize a lookup array with all initial values as NIL. Whenever we need solution to a subproblem, we first look into the lookup table. If the precomputed value is there then we return that value, otherwise we calculate the value and put the result in lookup table so that it can be reused later.

Following is the memoized version for nth Fibonacci Number.

```
/* C/C++ program for Memoized version for nth Fibonacci number */
#include<stdio.h>
#define NIL -1
#define MAX 100

int lookup[MAX];

/* Function to initialize NIL values in lookup table */
void _initialize()
{
    int i;
    for (i = 0; i < MAX; i++)
        lookup[i] = NIL;
}

/* function for nth Fibonacci number */
int fib(int n)
{
    if (lookup[n] == NIL)
    {
        if (n <= 1)
            lookup[n] = n;
        else
            lookup[n] = fib(n-1) + fib(n-2);
    }
    return lookup[n];
}

int main ()
{
    int n = 40;
    _initialize();
    printf("Fibonacci number is %d ", fib(n));
    return 0;
}
```

Run on IDE

Python

```
# Python program for Memoized version of nth Fibonacci number

# Function to calculate nth Fibonacci number
```

```

def fib(n, lookup):
    # Base case
    if n == 0 or n == 1 :
        lookup[n] = n

    # If the value is not calculated previously then calculate it
    if lookup[n] is None:
        lookup[n] = fib(n-1 , lookup) + fib(n-2 , lookup)

    # return the value corresponding to that value of n
    return lookup[n]
# end of function

# Driver program to test the above function
def main():
    n = 34
    # Declaration of lookup table
    # Handles till n = 100
    lookup = [None]*(101)
    print "Fibonacci Number is ", fib(n, lookup)

if __name__=="__main__":
    main()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)

```

Run on IDE

b) Tabulation (Bottom Up): The tabulated program for a given problem builds a table in bottom up fashion and returns the last entry from table. For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3) and so on. So literally, we are building the solutions of subproblems bottom-up.

Following is the tabulated version for nth Fibonacci Number.

```

/* C program for Tabulated version */
#include<stdio.h>
int fib(int n)
{
    int f[n+1];
    int i;
    f[0] = 0;   f[1] = 1;
    for (i = 2; i <= n; i++)
        f[i] = f[i-1] + f[i-2];

    return f[n];
}

int main ()
{
    int n = 9;
    printf("Fibonacci number is %d ", fib(n));
    return 0;
}

```

Run on IDE

Python

```

# Python program Tabulated (bottom up) version
def fib(n):
    # array declaration

```

```

f = [0]*(n+1)

# base case assignment
f[1] = 1

# calculating the fibonacci and storing the values
for i in xrange(2 , n+1):
    f[i] = f[i-1] + f[i-2]
return f[n]

# Driver program to test the above function
def main():
    n = 9
    print "Fibonacci number is " , fib(n)

if __name__=="__main__":
    main()

# This code is contributed by Nikhil Kumar Singh (nickzuck_007)

```

[Run on IDE](#)

Output:

Fibonacci number is 34

Both Tabulated and Memoized store the solutions of subproblems. In Memoized version, table is filled on demand while in Tabulated version, starting from the first entry, all entries are filled one by one. Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. For example, [Memoized solution](#) of the [LCS problem](#) doesn't necessarily fill all entries.

To see the optimization achieved by Memoized and Tabulated solutions over the basic Recursive solution, see the time taken by following runs for calculating 40th Fibonacci number:

[Recursive solution](#)

[Memoized solution](#)

[Tabulated solution](#)

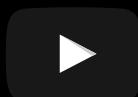
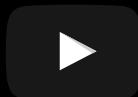
Time taken by Recursion method is much more than the two Dynamic Programming techniques mentioned above – Memoization and Tabulation!

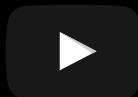
Also see method 2 of [Ugly Number post](#) for one more simple example where we have overlapping subproblems and we store the results of subproblems.

We will be covering Optimal Substructure Property and some more example problems in future posts on Dynamic Programming.

Try following questions as an exercise of this post.

- 1) Write a Memoized solution for LCS problem. Note that the Tabular solution is given in the CLRS book.
- 2) How would you choose between Memoization and Tabulation?

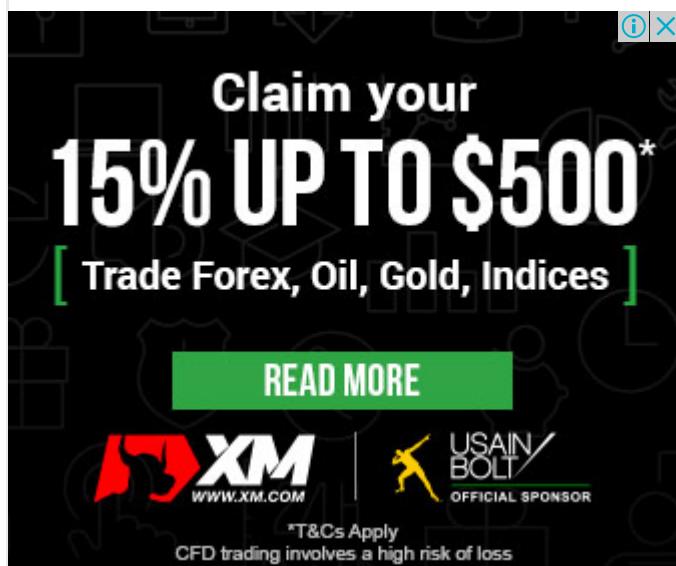




Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

<http://www.youtube.com/watch?v=V5hZoJ6uK-s> 



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [Fibonacci](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix

- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

1.5

Average Difficulty : **1.5/5.0**
Based on **129** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 2 (Optimal Substructure Property)

As we discussed in [Set 1](#), following are the two main properties of a problem that suggest that the given problem can be solved using Dynamic programming:

- 1) Overlapping Subproblems
- 2) Optimal Substructure

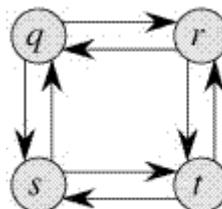
We have already discussed Overlapping Subproblem property in the [Set 1](#). Let us discuss Optimal Substructure property here.

2) Optimal Substructure: A given problems has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems.

For example, the Shortest Path problem has following optimal substructure property:

If a node x lies in the shortest path from a source node u to destination node v then the shortest path from u to v is combination of shortest path from u to x and shortest path from x to v. The standard All Pair Shortest Path algorithms like [Floyd–Warshall](#) and [Bellman–Ford](#) are typical examples of Dynamic Programming.

On the other hand, the Longest Path problem doesn't have the Optimal Substructure property. Here by Longest Path we mean longest simple path (path without cycle) between two nodes. Consider the following unweighted graph given in the [CLRS book](#). There are two longest paths from q to t: $q \rightarrow r \rightarrow t$ and $q \rightarrow s \rightarrow t$. Unlike shortest paths, these longest paths do not have the optimal substructure property. For example, the longest path $q \rightarrow r \rightarrow t$ is not a combination of longest path from q to r and longest path from r to t, because the longest path from q to r is $q \rightarrow s \rightarrow t \rightarrow r$ and the longest path from r to t is $r \rightarrow q \rightarrow s \rightarrow t$.



We will be covering some example problems in future posts on [Dynamic Programming](#).



Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

http://en.wikipedia.org/wiki/Optimal_substructure

CLRS book



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +

- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

1.6

Average Difficulty : **1.6/5.0**
Based on **91** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 3 (Longest Increasing Subsequence)

We have discussed Overlapping Subproblems and Optimal Substructure properties in [Set 1](#) and [Set 2](#) respectively.

Let us discuss Longest Increasing Subsequence (LIS) problem as an example problem that can be solved using Dynamic Programming.

The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.

arr[]	10	22	9	33	21	50	41	60	80
LIS	1	2		3		4		5	6

More Examples:

```
Input : arr[] = {3, 10, 2, 1, 20}
Output : Length of LIS = 3
The longest increasing subsequence is 3, 10, 20
```

```
Input : arr[] = {3, 2}
Output : Length of LIS = 1
The longest increasing subsequences are {3} and {2}
```

```
Input : arr[] = {50, 3, 10, 7, 40, 80}
Output : Length of LIS = 4
The longest increasing subsequence is {3, 7, 40, 80}
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Optimal Substructure:

Let $\text{arr}[0..n-1]$ be the input array and $L(i)$ be the length of the LIS ending at index i such that $\text{arr}[i]$ is the last element of the LIS.

Then, $L(i)$ can be recursively written as:

$L(i) = 1 + \max(L(j))$ where $0 < j < i$ and $\text{arr}[j] < \text{arr}[i]$; or

$L(i) = 1$, if no such j exists.

To find the LIS for a given array, we need to return $\max(L(i))$ where $0 < i < n$.

Thus, we see the LIS problem satisfies the optimal substructure property as the main problem can be solved using solutions to subproblems.

Following is a simple recursive implementation of the LIS problem. It follows the recursive structure discussed above.

```
// A naive C/C++ based recursive implementation of LIS problem
#include<stdio.h>
#include<stdlib.h>

// Recursive implementation for calculating the LIS
int _lis(int arr[], int n, int *max_lis_length)
{
    // Base case
    if (n == 1)
        return 1;

    int current_lis_length = 1;
    for (int i=0; i<n-1; i++)
    {
        // Recursively calculate the length of the LIS
        // ending at arr[i]
        int subproblem_lis_length = _lis(arr, i, max_lis_length);

        // Check if appending arr[n-1] to the LIS
        // ending at arr[i] gives us an LIS ending at
        // arr[n-1] which is longer than the previously
        // calculated LIS ending at arr[n-1]
        if (arr[i] < arr[n-1] &&
            current_lis_length < (1+subproblem_lis_length))
            current_lis_length = 1+subproblem_lis_length;
    }

    // Check if currently calculated LIS ending at
    // arr[n-1] is longer than the previously calculated
    // LIS and update max_lis_length accordingly
    if (*max_lis_length < current_lis_length)
        *max_lis_length = current_lis_length;

    return current_lis_length;
}

// The wrapper function for _lis()
int lis(int arr[], int n)
{
    int max_lis_length = 1; // stores the final LIS

    // max_lis_length is passed as a reference below
    // so that it can maintain its value
    // between the recursive calls
    _lis( arr, n, &max_lis_length );

    return max_lis_length;
}

// Driver program to test the functions above
int main()
{
    int arr[] = {10, 22, 9, 33, 21, 50, 41, 60};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Length of LIS is %d\n", lis( arr, n ));
    return 0;
}
```

Run on IDE

Java

```
// A naive Java based recursive implementation of LIS problem
class LIS
{
    static int max_lis_length; // stores the final LIS

    // Recursive implementation for calculating the LIS
    static int _lis(int arr[], int n)
    {
        // base case
        if (n == 1)
            return 1;

        int current_lis_length = 1;
        for (int i=0; i<n-1; i++)
        {
            // Recursively calculate the length of the LIS
            // ending at arr[i]
            int subproblem_lis_length = _lis(arr, i);

            // Check if appending arr[n-1] to the LIS
            // ending at arr[i] gives us an LIS ending at
            // arr[n-1] which is longer than the previously
            // calculated LIS ending at arr[n-1]
            if (arr[i] < arr[n-1] &&
                current_lis_length < (1+subproblem_lis_length))
                current_lis_length = 1+subproblem_lis_length;
        }

        // Check if currently calculated LIS ending at
        // arr[n-1] is longer than the previously calculated
        // LIS and update max_lis_length accordingly
        if (max_lis_length < current_lis_length)
            max_lis_length = current_lis_length;

        return current_lis_length;
    }

    // The wrapper function for _lis()
    static int lis(int arr[], int n)
    {
        max_lis_length = 1; // stores the final LIS

        // max_lis_length is declared static above
        // so that it can maintain its value
        // between the recursive calls of _lis()
        _lis( arr, n );

        return max_lis_length;
    }

    // Driver program to test the functions above
    public static void main(String args[])
    {
        int arr[] = {10, 22, 9, 33, 21, 50, 41, 60};
        int n = arr.length;
        System.out.println("Length of LIS is " + lis( arr, n ));
    }
}

} // End of LIS class.

// This code is contributed by Rajat Mishra
```

Run on IDE

Python

```
# A naive Python based recursive implementation of LIS problem
```

```

global max_lis_length # stores the final LIS

# Recursive implementation for calculating the LIS
def _lis(arr, n):
    # Following declaration is needed to allow modification
    # of the global copy of max_lis_length in _lis()
    global max_lis_length

    # Base Case
    if n == 1:
        return 1

    current_lis_length = 1

    for i in xrange(0, n-1):
        # Recursively calculate the length of the LIS
        # ending at arr[i]
        subproblem_lis_length = _lis(arr, i)

        # Check if appending arr[n-1] to the LIS
        # ending at arr[i] gives us an LIS ending at
        # arr[n-1] which is longer than the previously
        # calculated LIS ending at arr[n-1]
        if arr[i] < arr[n-1] and \
            current_lis_length < (1+subproblem_lis_length):
            current_lis_length = (1+subproblem_lis_length)

    # Check if currently calculated LIS ending at
    # arr[n-1] is longer than the previously calculated
    # LIS and update max_lis_length accordingly
    if (max_lis_length < current_lis_length):
        max_lis_length = current_lis_length

    return current_lis_length

# The wrapper function for _lis()
def lis(arr, n):

    # Following declaration is needed to allow modification
    # of the global copy of max_lis_length in lis()
    global max_lis_length

    max_lis_length = 1 # stores the final LIS

    # max_lis_length is declared global at the top
    # so that it can maintain its value
    # between the recursive calls of _lis()
    _lis(arr, n)

    return max_lis_length

# Driver program to test the functions above
def main():
    arr = [10, 22, 9, 33, 21, 50, 41, 60]
    n = len(arr)
    print "Length of LIS is", lis(arr, n)

if __name__=="__main__":
    main()

# This code is contributed by NIKHIL KUMAR SINGH

```

Run on IDE

Output:

Length of LIS is 5

Overlapping Subproblems:

Considering the above implementation, following is recursion tree for an array of size 4. lis(n) gives us the length of LIS for arr[].

```

        lis(4)
        /   |   \
lis(3)   lis(2)   lis(1)
 /   \   /
lis(2) lis(1) lis(1)
/
lis(1)

```

We can see that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation. Following is a tabulated implementation for the LIS problem.

```

/* Dynamic Programming C/C++ implementation of LIS problem */
#include<stdio.h>
#include<stdlib.h>

/* lis() returns the length of the longest increasing
   subsequence in arr[] of size n */
int lis( int arr[], int n )
{
    int *lis, i, j, max = 0;
    lis = (int*) malloc( sizeof( int ) * n );

    /* Initialize LIS values for all indexes */
    for (i = 0; i < n; i++)
        lis[i] = 1;

    /* Compute optimized LIS values in bottom up manner */
    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
            if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)
                lis[i] = lis[j] + 1;

    /* Pick maximum of all LIS values */
    for (i = 0; i < n; i++)
        if (max < lis[i])
            max = lis[i];

    /* Free memory to avoid memory leak */
    free(lis);
}

return max;
}

/* Driver program to test above function */
int main()
{
    int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Length of lis is %d\n", lis( arr, n ) );
    return 0;
}

```

Run on IDE

Java

```
/* Dynamic Programming Java implementation of LIS problem */

class LIS
{
    /* lis() returns the length of the longest increasing
       subsequence in arr[] of size n */
    static int lis(int arr[],int n)
    {
        int lis[] = new int[n];
        int i,j,max = 0;

        /* Initialize LIS values for all indexes */
        for ( i = 0; i < n; i++ )
            lis[i] = 1;

        /* Compute optimized LIS values in bottom up manner */
        for ( i = 1; i < n; i++ )
            for ( j = 0; j < i; j++ )
                if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)
                    lis[i] = lis[j] + 1;

        /* Pick maximum of all LIS values */
        for ( i = 0; i < n; i++ )
            if ( max < lis[i] )
                max = lis[i];

        return max;
    }

    public static void main(String args[])
    {
        int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
        int n = arr.length;
        System.out.println("Length of lis is " + lis( arr, n ) + "\n");
    }
}
/*This code is contributed by Rajat Mishra*/
```

[Run on IDE](#)

Python

```
# Dynamic programming Python implementation of LIS problem

# lis returns length of the longest increasing subsequence
# in arr of size n
def lis(arr):
    n = len(arr)

    # Declare the list (array) for LIS and initialize LIS
    # values for all indexes
    lis = [1]*n

    # Compute optimized LIS values in bottom up manner
    for i in range (1 , n):
        for j in range(0 , i):
            if arr[i] > arr[j] and lis[i]< lis[j] + 1 :
                lis[i] = lis[j]+1

    # Initialize maximum to 0 to get the maximum of all
    # LIS
    maximum = 0

    # Pick maximum of all LIS values
    for i in range(n):
        maximum = max(maximum , lis[i])

    return maximum
# end of lis function

# Driver program to test above function
```

```
arr = [10, 22, 9, 33, 21, 50, 41, 60]
print "Length of lis is", lis(arr)
# This code is contributed by Nikhil Kumar Singh
```

[Run on IDE](#)

Output:

```
Length of lis is 5
```

Note that the time complexity of the above Dynamic Programming (DP) solution is $O(n^2)$ and there is a $O(n \log n)$ solution for the LIS problem. We have not discussed the $O(n \log n)$ solution here as the purpose of this post is to explain Dynamic Programming with a simple example. See below post for $O(n \log n)$ solution.

Longest Increasing Subsequence Size ($N \log N$)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#)
[Dynamic Programming](#)
[LIS](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.1

Average Difficulty : **3.1/5.0**
Based on **201** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 4 (Longest Common Subsequence)

We have discussed Overlapping Subproblems and Optimal Substructure properties in [Set 1](#) and [Set 2](#) respectively. We also discussed one example problem in [Set 3](#). Let us discuss Longest Common Subsequence (LCS) problem as one more example problem that can be solved using Dynamic Programming.

LCS Problem Statement: Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, "abc", "abg", "bdf", "aeg", "acefg", .. etc are subsequences of "abcdefg". So a string of length n has 2^n different possible subsequences.

It is a classic computer science problem, the basis of [diff](#) (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.

Examples:

LCS for input Sequences "ABCDGH" and "AEDFHR" is "ADH" of length 3.

LCS for input Sequences "AGGTAB" and "GXTXAYB" is "GTAB" of length 4.

We strongly recommend that you click here and practice it, before moving on to the solution.

The naive solution for this problem is to generate all subsequences of both given sequences and find the longest matching subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem.

1) Optimal Substructure:

Let the input sequences be $X[0..m-1]$ and $Y[0..n-1]$ of lengths m and n respectively. And let $L(X[0..m-1], Y[0..n-1])$ be the length of LCS of the two sequences X and Y. Following is the recursive definition of $L(X[0..m-1], Y[0..n-1])$.

If last characters of both sequences match (or $X[m-1] == Y[n-1]$) then

$$L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])$$

If last characters of both sequences do not match (or $X[m-1] != Y[n-1]$) then

$$L(X[0..m-1], Y[0..n-1]) = \text{MAX} (L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2]))$$

Examples:

1) Consider the input strings "AGGTAB" and "GXTXAYB". Last characters match for the strings. So length of LCS

can be written as:

$$L("AGGTAB", "GXTXAYB") = 1 + L("AGGTA", "GXTXAY")$$

	A	G	G	T	A	B
G	-	-	4	-	-	-
X	-	-	-	-	-	-
T	-	-	-	3	-	-
X	-	-	-	-	-	-
A	-	-	-	-	2	-
Y	-	-	-	-	-	-
B	-	-	-	-	-	1

2) Consider the input strings “ABCDGH” and “AEDFHR”. Last characters do not match for the strings. So length of LCS can be written as:

$$L("ABCDGH", "AEDFHR") = \text{MAX} (L("ABCDG", "AEDFHR"), L("ABCDGH", "AEDFH"))$$

So the LCS problem has optimal substructure property as the main problem can be solved using solutions to subproblems.

2) Overlapping Subproblems:

Following is simple recursive implementation of the LCS problem. The implementation simply follows the recursive structure mentioned above.

```
/* A Naive recursive implementation of LCS problem */
#include<bits/stdc++.h>

int max(int a, int b);

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m-1] == Y[n-1])
        return 1 + lcs(X, Y, m-1, n-1);
    else
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}

/* Utility function to get max of 2 integers */
int max(int a, int b)
{
    return (a > b)? a : b;
}

/* Driver program to test above function */
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);

    printf("Length of LCS is %d\n", lcs( X, Y, m, n ) );
}
```

```

    return 0;
}

```

[Run on IDE](#)

Python

```

# A Naive recursive Python implementation of LCS problem

def lcs(X, Y, m, n):

    if m == 0 or n == 0:
        return 0;
    elif X[m-1] == Y[n-1]:
        return 1 + lcs(X, Y, m-1, n-1);
    else:
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));

# Driver program to test the above function
X = "AGGTAB"
Y = "GXTXAYB"
print "Length of LCS is ", lcs(X , Y, len(X), len(Y))

```

[Run on IDE](#)

Output:

Length of LCS is 4

Time complexity of the above naive recursive approach is $O(2^n)$ in worst case and worst case happens when all characters of X and Y mismatch i.e., length of LCS is 0.

Considering the above implementation, following is a partial recursion tree for input strings “AXYT” and “AYZX”

```

          lcs("AXYT", "AYZX")
          /           \
lcs("AXY", "AYZX")      lcs("AXYT", "AYZ")
 /       \           /   \
lcs("AX", "AYZX") lcs("AXY", "AYZ") lcs("AXY", "AYZ") lcs("AXYT", "AY")

```

In the above partial recursion tree, $\text{lcs}(\text{“AXY”, “AYZ”})$ is being solved twice. If we draw the complete recursion tree, then we can see that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation. Following is a tabulated implementation for the LCS problem.

```

/* Dynamic Programming C/C++ implementation of LCS problem */
#include<bits/stdc++.h>

int max(int a, int b);

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
    int L[m+1][n+1];
    int i, j;

```

```

/* Following steps build L[m+1][n+1] in bottom up fashion. Note
   that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
for (i=0; i<=m; i++)
{
    for (j=0; j<=n; j++)
    {
        if (i == 0 || j == 0)
            L[i][j] = 0;

        else if (X[i-1] == Y[j-1])
            L[i][j] = L[i-1][j-1] + 1;

        else
            L[i][j] = max(L[i-1][j], L[i][j-1]);
    }
}

/* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] */
return L[m][n];
}

/* Utility function to get max of 2 integers */
int max(int a, int b)
{
    return (a > b)? a : b;
}

/* Driver program to test above function */
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);

    printf("Length of LCS is %d\n", lcs( X, Y, m, n ) );
    return 0;
}

```

[Run on IDE](#)

Python

```

# Dynamic Programming implementation of LCS problem

def lcs(X , Y):
    # find the length of the strings
    m = len(X)
    n = len(Y)

    # declaring the array for storing the dp values
    L = [[None]*(n+1) for i in xrange(m+1)]

    """Following steps build L[m+1][n+1] in bottom up fashion
    Note: L[i][j] contains length of LCS of X[0..i-1]
    and Y[0..j-1]"""
    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0 :
                L[i][j] = 0
            elif X[i-1] == Y[j-1]:
                L[i][j] = L[i-1][j-1]+1
            else:
                L[i][j] = max(L[i-1][j] , L[i][j-1])

    # L[m][n] contains the length of LCS of X[0..n-1] & Y[0..m-1]
    return L[m][n]
#end of function lcs

```

```
# Driver program to test the above function
X = "AGGTAB"
Y = "GXTXAYB"
print "Length of LCS is ", lcs(X, Y)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

[Run on IDE](#)

Time Complexity of the above implementation is $O(mn)$ which is much better than the worst case time complexity of Naive Recursive implementation.

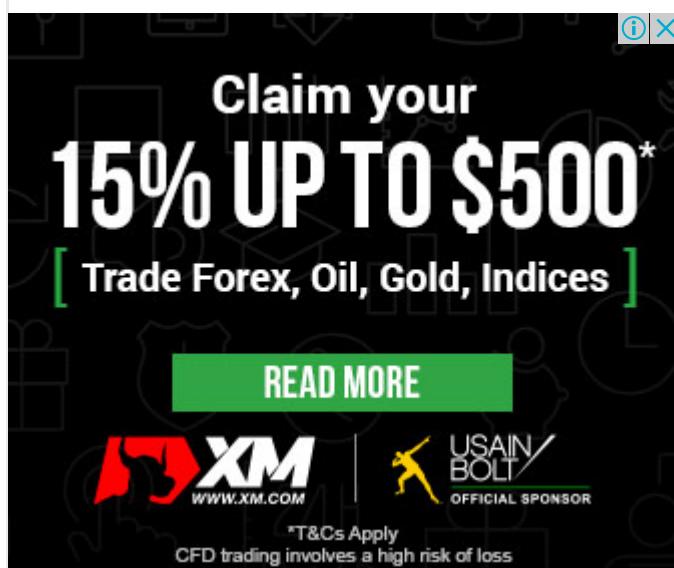
The above algorithm/code returns only length of LCS. Please see the following post for printing the LCS.

Printing Longest Common Subsequence

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

<http://www.youtube.com/watch?v=V5hZoJ6uK-s> 
http://www.algorithmist.com/index.php/Longest_Common_Subsequence
<http://www.ics.uci.edu/~eppstein/161/960229.html>
http://en.wikipedia.org/wiki/Longest_common_subsequence_problem



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [LCS](#) [subsequence](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix

- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.7

Average Difficulty : 2.7/5.0
Based on 173 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 5 (Edit Distance)

Given two strings str1 and str2 and below operations that can be performed on str1. Find minimum number of edits (operations) required to convert 'str1' into 'str2'.

- a. Insert
- b. Remove
- c. Replace

All of the above operations are of equal cost.

Examples:

```
Input: str1 = "geek", str2 = "gesek"
Output: 1
We can convert str1 into str2 by inserting a 's'.
```

```
Input: str1 = "cat", str2 = "cut"
Output: 1
We can convert str1 into str2 by replacing 'a' with 'u'.
```

```
Input: str1 = "sunday", str2 = "saturday"
Output: 3
Last three and first characters are same. We basically
need to convert "un" to "atur". This can be done using
below three operations.
Replace 'n' with 'r', insert t, insert a
```

We strongly recommend that you click here and practice it, before moving on to the solution.

What are the subproblems in this case?

The idea is process all characters one by one starting from either from left or right sides of both strings.

Let's traverse from right corner, there are two possibilities for every pair of character being traversed.

```
m: Length of str1 (first string)
n: Length of str2 (second string)
```

1. If last characters of two strings are same, nothing much to do. Ignore last characters and get count for remaining strings. So we recur for lengths m-1 and n-1.
2. Else (If last characters are not same), we consider all operations on 'str1', consider all three operations on last character of first string, recursively compute minimum cost for all three operations and take minimum of three values.
 - a. Insert: Recur for m and n-1
 - b. Remove: Recur for m-1 and n
 - c. Replace: Recur for m-1 and n-1

Below is C++ implementation of above Naive recursive solution.

```
// A Naive recursive C++ program to find minimum number
// operations to convert str1 to str2
#include<bits/stdc++.h>
using namespace std;

// Utility function to find minimum of three numbers
int min(int x, int y, int z)
{
    return min(min(x, y), z);
}

int editDist(string str1 , string str2 , int m ,int n)
{
    // If first string is empty, the only option is to
    // insert all characters of second string into first
    if (m == 0) return n;

    // If second string is empty, the only option is to
    // remove all characters of first string
    if (n == 0) return m;

    // If last characters of two strings are same, nothing
    // much to do. Ignore last characters and get count for
    // remaining strings.
    if (str1[m-1] == str2[n-1])
        return editDist(str1, str2, m-1, n-1);

    // If last characters are not same, consider all three
    // operations on last character of first string, recursively
    // compute minimum cost for all three operations and take
    // minimum of three values.
    return 1 + min ( editDist(str1, str2, m, n-1),    // Insert
                     editDist(str1, str2, m-1, n),    // Remove
                     editDist(str1, str2, m-1, n-1) // Replace
                   );
}

// Driver program
int main()
{
    // your code goes here
    string str1 = "sunday";
    string str2 = "saturday";

    cout << editDist( str1 , str2 , str1.length(), str2.length());

    return 0;
}
```

Run on IDE

Java

```
// A Naive recursive Java program to find minimum number
// operations to convert str1 to str2
class EDIST
{
    static int min(int x,int y,int z)
    {
        if (x<y && x<z) return x;
        if (y<x && y<z) return y;
        else return z;
    }

    static int editDist(String str1 , String str2 , int m ,int n)
    {
        // If first string is empty, the only option is to
        // insert all characters of second string into first
        if (m == 0) return n;

        // If second string is empty, the only option is to
        // remove all characters of first string
        if (n == 0) return m;

        // If last characters of two strings are same, nothing
        // much to do. Ignore last characters and get count for
        // remaining strings.
        if (str1.charAt(m-1) == str2.charAt(n-1))
            return editDist(str1, str2, m-1, n-1);

        // If last characters are not same, consider all three
        // operations on last character of first string, recursively
        // compute minimum cost for all three operations and take
        // minimum of three values.
        return 1 + min ( editDist(str1, str2, m, n-1),      // Insert
                        editDist(str1, str2, m-1, n),      // Remove
                        editDist(str1, str2, m-1, n-1) // Replace
                      );
    }

    public static void main(String args[])
    {
        String str1 = "sunday";
        String str2 = "saturday";

        System.out.println( editDist( str1 , str2 , str1.length(), str2.length()) );
    }
}
/*This code is contributed by Rajat Mishra*/
```

Run on IDE

Python

```
# A Naive recursive Python program to fin minimum number
# operations to convert str1 to str2
def editDistance(str1, str2, m , n):

    # If first string is empty, the only option is to
    # insert all characters of second string into first
    if m==0:
        return n

    # If second string is empty, the only option is to
    # remove all characters of first string
    if n==0:
        return m

    # If last characters of two strings are same, nothing
    # much to do. Ignore last characters and get count for
```

```

# remaining strings.
if str1[m-1]==str2[n-1]:
    return editDistance(str1,str2,m-1,n-1)

# If last characters are not same, consider all three
# operations on last character of first string, recursively
# compute minimum cost for all three operations and take
# minimum of three values.
return 1 + min(editDistance(str1, str2, m, n-1),      # Insert
               editDistance(str1, str2, m-1, n),      # Remove
               editDistance(str1, str2, m-1, n-1))     # Replace
)

# Driver program to test the above function
str1 = "sunday"
str2 = "saturday"
print editDistance(str1, str2, len(str1), len(str2))

# This code is contributed by Bhavya Jain

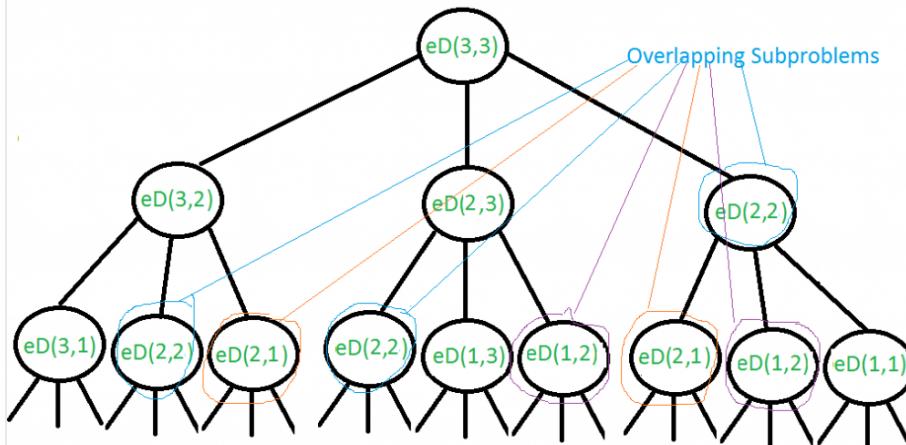
```

[Run on IDE](#)

Output:

3

The time complexity of above solution is exponential. In worst case, we may end up doing $O(3^m)$ operations. The worst case happens when none of characters of two strings match. Below is a recursive call diagram for worst case.



Worst case recursion tree when $m = 3, n = 3$.
Worst case example str1="abc" str2="xyz"

We can see that many subproblems are solved again and again, for example $eD(2,2)$ is called three times. Since same subproblems are called again, this problem has Overlapping Subproblems property. So Edit Distance problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array that stores results of subproblems.

```

// A Dynamic Programming based C++ program to find minimum
// number operations to convert str1 to str2
#include<bits/stdc++.h>
using namespace std;

```

```

// Utility function to find minimum of three numbers
int min(int x, int y, int z)
{
    return min(min(x, y), z);
}

int editDistDP(string str1, string str2, int m, int n)
{
    // Create a table to store results of subproblems
    int dp[m+1][n+1];

    // Fill d[][] in bottom up manner
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            // If first string is empty, only option is to
            // insert all characters of second string
            if (i==0)
                dp[i][j] = j; // Min. operations = j

            // If second string is empty, only option is to
            // remove all characters of second string
            else if (j==0)
                dp[i][j] = i; // Min. operations = i

            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1];

            // If last character are different, consider all
            // possibilities and find minimum
            else
                dp[i][j] = 1 + min(dp[i][j-1], // Insert
                                    dp[i-1][j], // Remove
                                    dp[i-1][j-1]); // Replace
        }
    }

    return dp[m][n];
}

```

```

// Driver program
int main()
{
    // your code goes here
    string str1 = "sunday";
    string str2 = "saturday";

    cout << editDistDP(str1, str2, str1.length(), str2.length());

    return 0;
}

```

Run on IDE

Java

```

// A Dynamic Programming based Java program to find minimum
// number operations to convert str1 to str2
class EDIST
{
    static int min(int x,int y,int z)
    {
        if (x < y && x < z) return x;
        if (y < x && y < z) return y;
        else return z;
    }
}

```

```

static int editDistDP(String str1, String str2, int m, int n)
{
    // Create a table to store results of subproblems
    int dp[][] = new int[m+1][n+1];

    // Fill d[][] in bottom up manner
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            // If first string is empty, only option is to
            // insert all characters of second string
            if (i==0)
                dp[i][j] = j; // Min. operations = j

            // If second string is empty, only option is to
            // remove all characters of second string
            else if (j==0)
                dp[i][j] = i; // Min. operations = i

            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1.charAt(i-1) == str2.charAt(j-1))
                dp[i][j] = dp[i-1][j-1];

            // If last character are different, consider all
            // possibilities and find minimum
            else
                dp[i][j] = 1 + min(dp[i][j-1], // Insert
                                   dp[i-1][j], // Remove
                                   dp[i-1][j-1]); // Replace
        }
    }

    return dp[m][n];
}

```

```

public static void main(String args[])
{
    String str1 = "sunday";
    String str2 = "saturday";
    System.out.println( editDistDP( str1 , str2 , str1.length(), str2.length() ) );
}

/*This code is contributed by Rajat Mishra*/

```

[Run on IDE](#)

Python

```

# A Dynamic Programming based Python program for edit
# distance problem
def editDistDP(str1, str2, m, n):
    # Create a table to store results of subproblems
    dp = [[0 for x in range(n+1)] for x in range(m+1)]

    # Fill d[][] in bottom up manner
    for i in range(m+1):
        for j in range(n+1):

            # If first string is empty, only option is to
            # insert all characters of second string
            if i == 0:
                dp[i][j] = j # Min. operations = j

            # If second string is empty, only option is to
            # remove all characters of second string
            elif j == 0:
                dp[i][j] = i # Min. operations = i

```

```

# If last characters are same, ignore last char
# and recur for remaining string
elif str1[i-1] == str2[j-1]:
    dp[i][j] = dp[i-1][j-1]

# If last character are different, consider all
# possibilities and find minimum
else:
    dp[i][j] = 1 + min(dp[i][j-1],           # Insert
                        dp[i-1][j],          # Remove
                        dp[i-1][j-1])        # Replace

return dp[m][n]

# Driver program
str1 = "sunday"
str2 = "saturday"

print(editDistDP(str1, str2, len(str1), len(str2)))
# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

3

Time Complexity: $O(m \times n)$

Auxiliary Space: $O(m \times n)$

Applications: There are many practical applications of edit distance algorithm, refer [Lucene API](#) for sample. Another example, display all the words in a dictionary that are near proximity to a given word\incorrectly spelled word.

Asked in: Amazon

Thanks to Vivek Kumar for suggesting above updates.

Thanks to **Venki** for providing initial post. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

About Venki

Software Engineer

[View all posts by Venki →](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 138 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 6 (Min Cost Path)

Given a cost matrix $\text{cost}[][]$ and a position (m, n) in $\text{cost}[][]$, write a function that returns cost of minimum cost path to reach (m, n) from $(0, 0)$. Each cell of the matrix represents a cost to traverse through that cell. Total cost of a path to reach (m, n) is sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j) , cells $(i+1, j)$, $(i, j+1)$ and $(i+1, j+1)$ can be traversed. You may assume that all costs are positive integers.

For example, in the following figure, what is the minimum cost path to $(2, 2)$?

1	2	3
4	8	2
1	5	3

The path with minimum cost is highlighted in the following figure. The path is $(0, 0) \rightarrow (0, 1) \rightarrow (1, 2) \rightarrow (2, 2)$. The cost of the path is 8 ($1 + 2 + 2 + 3$).

1	2	3
4	8	2
1	5	3

1) Optimal Substructure

The path to reach (m, n) must be through one of the 3 cells: $(m-1, n-1)$ or $(m-1, n)$ or $(m, n-1)$. So minimum cost to reach (m, n) can be written as “minimum of the 3 cells plus $\text{cost}[m][n]$ ”.

$$\text{minCost}(m, n) = \min(\text{minCost}(m-1, n-1), \text{minCost}(m-1, n), \text{minCost}(m, n-1)) + \text{cost}[m][n]$$

2) Overlapping Subproblems

Following is simple recursive implementation of the MCP (Minimum Cost Path) problem. The implementation simply follows the recursive structure mentioned above.

```
/* A Naive recursive implementation of MCP(Minimum Cost Path) problem */
#include<stdio.h>
#include<limits.h>
```

```

#define R 3
#define C 3

int min(int x, int y, int z);

/* Returns cost of minimum cost path from (0,0) to (m, n) in mat[R][C]*/
int minCost(int cost[R][C], int m, int n)
{
    if (n < 0 || m < 0)
        return INT_MAX;
    else if (m == 0 && n == 0)
        return cost[m][n];
    else
        return cost[m][n] + min( minCost(cost, m-1, n-1),
                                minCost(cost, m-1, n),
                                minCost(cost, m, n-1) );
}

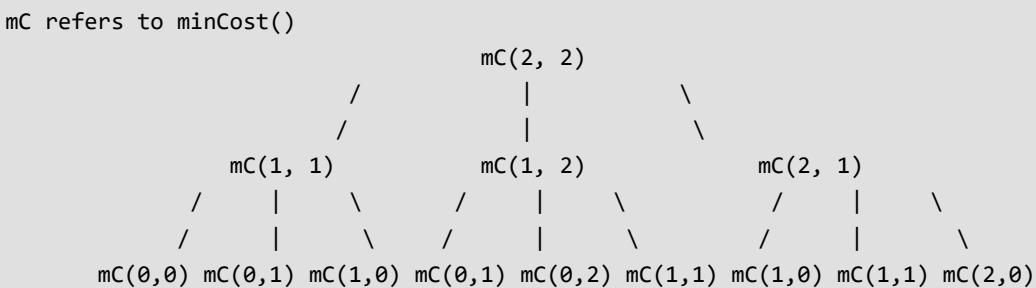
/* A utility function that returns minimum of 3 integers */
int min(int x, int y, int z)
{
    if (x < y)
        return (x < z)? x : z;
    else
        return (y < z)? y : z;
}

/* Driver program to test above functions */
int main()
{
    int cost[R][C] = { {1, 2, 3},
                      {4, 8, 2},
                      {1, 5, 3} };
    printf(" %d ", minCost(cost, 2, 2));
    return 0;
}

```

[Run on IDE](#)

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree, there are many nodes which appear more than once. Time complexity of this naive recursive solution is exponential and it is terribly slow.



So the MCP problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array `tc[][]` in bottom up manner.

```

/* Dynamic Programming implementation of MCP problem */
#include<stdio.h>
#include<limits.h>
#define R 3
#define C 3

int min(int x, int y, int z);

```

```

int minCost(int cost[R][C], int m, int n)
{
    int i, j;

    // Instead of following line, we can use int tc[m+1][n+1] or
    // dynamically allocate memory to save space. The following line is
    // used to keep the program simple and make it working on all compilers.
    int tc[R][C];

    tc[0][0] = cost[0][0];

    /* Initialize first column of total cost(tc) array */
    for (i = 1; i <= m; i++)
        tc[i][0] = tc[i-1][0] + cost[i][0];

    /* Initialize first row of tc array */
    for (j = 1; j <= n; j++)
        tc[0][j] = tc[0][j-1] + cost[0][j];

    /* Construct rest of the tc array */
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
            tc[i][j] = min(tc[i-1][j-1],
                            tc[i-1][j],
                            tc[i][j-1]) + cost[i][j];

    return tc[m][n];
}

/* A utility function that returns minimum of 3 integers */
int min(int x, int y, int z)
{
    if (x < y)
        return (x < z)? x : z;
    else
        return (y < z)? y : z;
}

/* Driver program to test above functions */
int main()
{
    int cost[R][C] = { {1, 2, 3},
                      {4, 8, 2},
                      {1, 5, 3} };
    printf(" %d ", minCost(cost, 2, 2));
    return 0;
}

```

[Run on IDE](#)

Java

```

/* Java program for Dynamic Programming implementation
   of Min Cost Path problem */
import java.util.*;

class MinimumCostPath
{
    /* A utility function that returns minimum of 3 integers */
    private static int min(int x, int y, int z)
    {
        if (x < y)
            return (x < z)? x : z;
        else
            return (y < z)? y : z;
    }

    private static int minCost(int cost[][][], int m, int n)
    {
        int i, j;

```

```

int tc[][]=new int[m+1][n+1];

tc[0][0] = cost[0][0];

/* Initialize first column of total cost(tc) array */
for (i = 1; i <= m; i++)
    tc[i][0] = tc[i-1][0] + cost[i][0];

/* Initialize first row of tc array */
for (j = 1; j <= n; j++)
    tc[0][j] = tc[0][j-1] + cost[0][j];

/* Construct rest of the tc array */
for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++)
        tc[i][j] = min(tc[i-1][j-1],
                        tc[i-1][j],
                        tc[i][j-1]) + cost[i][j];

return tc[m][n];
}

/* Driver program to test above functions */
public static void main(String args[])
{
    int cost[][]= {{1, 2, 3},
                  {4, 8, 2},
                  {1, 5, 3}};
    System.out.println("minimum cost to reach (2,2) = " +
                       minCost(cost,2,2));
}
// This code is contributed by Pankaj Kumar

```

[Run on IDE](#)

Python

```

# Dynamic Programming Python implementation of Min Cost Path
# problem
R = 3
C = 3

def minCost(cost, m, n):

    # Instead of following line, we can use int tc[m+1][n+1] or
    # dynamically allocate memory to save space. The following
    # line is used to keep the program simple and make it working
    # on all compilers.
    tc = [[0 for x in range(C)] for x in range(R)]

    tc[0][0] = cost[0][0]

    # Initialize first column of total cost(tc) array
    for i in range(1, m+1):
        tc[i][0] = tc[i-1][0] + cost[i][0]

    # Initialize first row of tc array
    for j in range(1, n+1):
        tc[0][j] = tc[0][j-1] + cost[0][j]

    # Construct rest of the tc array
    for i in range(1, m+1):
        for j in range(1, n+1):
            tc[i][j] = min(tc[i-1][j-1], tc[i-1][j], tc[i][j-1]) + cost[i][j]

    return tc[m][n]

# Driver program to test above functions
cost = [[1, 2, 3],
        [4, 8, 2],
        [1, 5, 3]]

```

```
[1, 5, 3]
print(minCost(cost, 2, 2))

# This code is contributed by Bhavya Jain
```

[Run on IDE](#)**Output:**

8

Time Complexity of the DP implementation is $O(mn)$ which is much better than Naive Recursive implementation.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Matrix](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

[\(Login to Rate and Mark\)](#)**2.3**Average Difficulty : **2.3/5.0**
Based on **133** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 7 (Coin Change)

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, … , Sm} valued coins, how many ways can we make the change? The order of coins doesn't matter.

For example, for N = 4 and S = {1,2,3}, there are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}. So output should be 4.

For N = 10 and S = {2, 5, 3, 6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So the output should be 5.

1) Optimal Substructure

To count total number solutions, we can divide all set solutions in two sets.

- 1) Solutions that do not contain mth coin (or Sm).
- 2) Solutions that contain at least one Sm.

Let count(S[], m, n) be the function to count the number of solutions, then it can be written as sum of count(S[], m-1, n) and count(S[], m, n-Sm).

Therefore, the problem has optimal substructure property as the problem can be solved using solutions to subproblems.

2) Overlapping Subproblems

Following is a simple recursive implementation of the Coin Change problem. The implementation simply follows the recursive structure mentioned above.

```
#include<stdio.h>

// Returns the count of ways we can sum S[0...m-1] coins to get sum n
int count( int S[], int m, int n )
{
    // If n is 0 then there is 1 solution (do not include any coin)
    if (n == 0)
        return 1;

    // If n is less than 0 then no solution exists
    if (n < 0)
        return 0;

    // If there are no coins and n is greater than 0, then no solution exist
    if (m <=0 && n >= 1)
        return 0;

    // count is sum of solutions (i) including S[m-1] (ii) excluding S[m-1]
    return count( S, m - 1, n ) + count( S, m, n-S[m-1] );
}

// Driver program to test above function
int main()
{
    int i, j;
    int arr[] = {1, 2, 3};
```

```
int m = sizeof(arr)/sizeof(arr[0]);
printf("%d ", count(arr, m, 4));
getchar();
return 0;
}
```

Run on IDE

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree for $S = \{1, 2, 3\}$ and $n = 5$.

The function $C(\{1\}, 3)$ is called two times. If we draw the complete tree, then we can see that there are many subproblems being called more than once.

```

C() --> count()
               C({1,2,3}, 5)
                  /           \
      C({1,2,3}, 2)       C({1,2}, 5)
         /   \           /   \
C({1,2,3}, -1) C({1,2}, 2)   C({1,2}, 3)   C({1}, 5)
            /   \           /   \
            /   \           /   \
C({1,2}, 0)  C({1}, 2)  C({1,2}, 1) C({1}, 3)  C({1}, 4)  C({}, 5)
               / \     / \     / \
               / \     / \     / \
               .   .   .   .   .   C({1}, 3) C({}, 4)
                                         / \
                                         / \

```

Since same subproblems are called again, this problem has Overlapping Subproblems property. So the Coin Change problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array table $[\cdot][\cdot]$ in bottom up manner.

Dynamic Programming Solution

```
#include<stdio.h>

int count( int S[], int m, int n )
{
    int i, j, x, y;

    // We need n+1 rows as the table is constructed in bottom up manner using
    // the base case 0 value case (n = 0)
    int table[n+1][m];

    // Fill the entries for 0 value case (n = 0)
    for (i=0; i<m; i++)
        table[0][i] = 1;

    // Fill rest of the table entries in bottom up manner
    for (i = 1; i < n+1; i++)
    {
        for (j = 0; j < m; j++)
        {
            // Count of solutions including S[j]
            x = (i-S[j] >= 0)? table[i - S[j]][j]: 0;
            table[i][j] = table[i-1][j] + x;
        }
    }
}
```

```

        // Count of solutions excluding S[j]
        y = (j >= 1)? table[i][j-1]: 0;

        // total count
        table[i][j] = x + y;
    }
}
return table[n][m-1];
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 2, 3};
    int m = sizeof(arr)/sizeof(arr[0]);
    int n = 4;
    printf(" %d ", count(arr, m, n));
    return 0;
}

```

[Run on IDE](#)

Java

```

/* Dynamic Programming Java implementation of Coin
   Change problem */
import java.util.Arrays;

class CoinChange
{
    static long countWays(int S[], int m, int n)
    {
        //Time complexity of this function: O(mn)
        //Space Complexity of this function: O(n)

        // table[i] will be storing the number of solutions
        // for value i. We need n+1 rows as the table is
        // constructed in bottom up manner using the base
        // case (n = 0)
        long[] table = new long[n+1];

        // Initialize all table values as 0
        Arrays.fill(table, 0);    //O(n)

        // Base case (If given value is 0)
        table[0] = 1;

        // Pick all coins one by one and update the table[]
        // values after the index greater than or equal to
        // the value of the picked coin
        for (int i=0; i<m; i++)
            for (int j=S[i]; j<=n; j++)
                table[j] += table[j-S[i]];

        return table[n];
    }

    // Driver Function to test above function
    public static void main(String args[])
    {
        int arr[] = {1, 2, 3};
        int m = arr.length;
        int n = 4;
        System.out.println(countWays(arr, m, n));
    }
}
// This code is contributed by Pankaj Kumar

```

[Run on IDE](#)

Python

```
# Dynamic Programming Python implementation of Coin Change problem
def count(S, m, n):
    # We need n+1 rows as the table is constructed in bottom up
    # manner using the base case 0 value case (n = 0)
    table = [[0 for x in range(m)] for x in range(n+1)]

    # Fill the entries for 0 value case (n = 0)
    for i in range(m):
        table[0][i] = 1

    # Fill rest of the table entries in bottom up manner
    for i in range(1, n+1):
        for j in range(m):
            # Count of solutions including S[j]
            x = table[i - S[j]][j] if i-S[j] >= 0 else 0

            # Count of solutions excluding S[j]
            y = table[i][j-1] if j >= 1 else 0

            # total count
            table[i][j] = x + y

    return table[n][m-1]

# Driver program to test above function
arr = [1, 2, 3]
m = len(arr)
n = 4
print(count(arr, m, n))

# This code is contributed by Bhavya Jain
```

[Run on IDE](#)

Output:

4

Time Complexity: O(mn)

Following is a simplified version of method 2. The auxiliary space required here is O(n) only.

```
int count( int S[], int m, int n )
{
    // table[i] will be storing the number of solutions for
    // value i. We need n+1 rows as the table is constructed
    // in bottom up manner using the base case (n = 0)
    int table[n+1];

    // Initialize all table values as 0
    memset(table, 0, sizeof(table));

    // Base case (If given value is 0)
    table[0] = 1;

    // Pick all coins one by one and update the table[] values
    // after the index greater than or equal to the value of the
    // picked coin
    for(int i=0; i<m; i++)
        for(int j=S[i]; j<=n; j++)
            table[j] += table[j-S[i]];
```

```

    return table[n];
}

```

[Run on IDE](#)

Python

```

# Dynamic Programming Python implementation of Coin
# Change problem
def count(S, m, n):

    # table[i] will be storing the number of solutions for
    # value i. We need n+1 rows as the table is constructed
    # in bottom up manner using the base case (n = 0)
    # Initialize all table values as 0
    table = [0 for k in range(n+1)]

    # Base case (If given value is 0)
    table[0] = 1

    # Pick all coins one by one and update the table[] values
    # after the index greater than or equal to the value of the
    # picked coin
    for i in range(0,m):
        for j in range(S[i],n+1):
            table[j] += table[j-S[i]]

    return table[n]

# Driver program to test above function
arr = [1, 2, 3]
m = len(arr)
n = 4
x = count(arr, m, n)
print (x)

# This code is contributed by Afzal Ansari

```

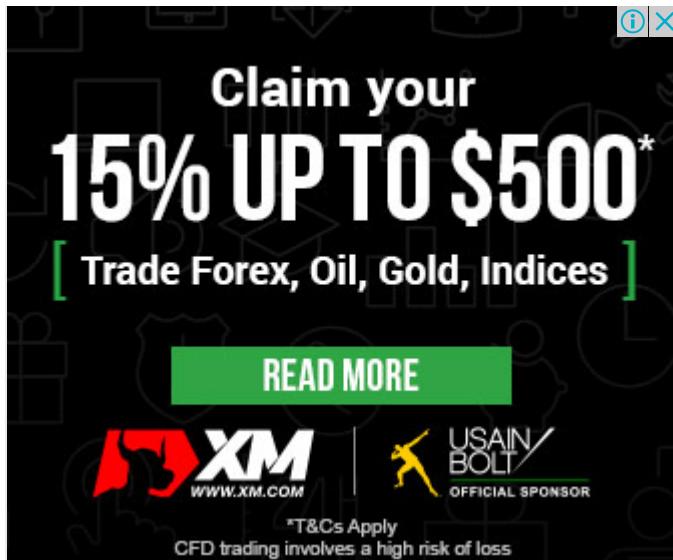
[Run on IDE](#)

Thanks to Rohan Laishram for suggesting this space optimized version.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

http://www.algorithmist.com/index.php/Coin_Change



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 163 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 8 (Matrix Chain Multiplication)

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have:

$$(ABC)D = (AB)(CD) = A(BCD) = \dots$$

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. For example, suppose A is a 10×30 matrix, B is a 30×5 matrix, and C is a 5×60 matrix. Then,

$$\begin{aligned} (AB)C &= (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ operations} \\ A(BC) &= (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ operations.} \end{aligned}$$

Clearly the first parenthesization requires less number of operations.

Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

Input: p[] = {40, 20, 30, 10, 30}

Output: 26000

There are 4 matrices of dimensions 40×20 , 20×30 , 30×10 and 10×30 .

Let the input 4 matrices be A, B, C and D. The minimum number of multiplications are obtained by putting parenthesis in following way

$$(A(BC))D \rightarrow 20 \times 30 \times 10 + 40 \times 20 \times 10 + 40 \times 10 \times 30$$

Input: p[] = {10, 20, 30, 40, 30}

Output: 30000

There are 4 matrices of dimensions 10×20 , 20×30 , 30×40 and 40×30 .

Let the input 4 matrices be A, B, C and D. The minimum number of multiplications are obtained by putting parenthesis in following way

$$((AB)C)D \rightarrow 10 \times 20 \times 30 + 10 \times 30 \times 40 + 10 \times 40 \times 30$$

Input: p[] = {10, 20, 30}

Output: 6000

There are only two matrices of dimensions 10x20 and 20x30. So there is only one way to multiply the matrices, cost of which is $10 \times 20 \times 30$

1) Optimal Substructure:

A simple solution is to place parenthesis at all possible places, calculate the cost for each placement and return the minimum value. In a chain of matrices of size n , we can place the first set of parenthesis in $n-1$ ways. For example, if the given chain is of 4 matrices. let the chain be ABCD, then there are 3 ways to place first set of parenthesis outer side: (A)(BCD), (AB)(CD) and (ABC)(D). So when we place a set of parenthesis, we divide the problem into subproblems of smaller size. Therefore, the problem has optimal substructure property and can be easily solved using recursion.

Minimum number of multiplication needed to multiply a chain of size n = Minimum of all $n-1$ placements (these placements create subproblems of smaller size)

2) Overlapping Subproblems

Following is a recursive implementation that simply follows the above optimal substructure property.

```
/* A naive recursive implementation that simply
   follows the above optimal substructure property */
#include<stdio.h>
#include<limits.h>

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
int MatrixChainOrder(int p[], int i, int j)
{
    if(i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;

    // place parenthesis at different places between first
    // and last matrix, recursively calculate count of
    // multiplications for each parenthesis placement and
    // return the minimum count
    for (k = i; k <j; k++)
    {
        count = MatrixChainOrder(p, i, k) +
            MatrixChainOrder(p, k+1, j) +
            p[i-1]*p[k]*p[j];

        if (count < min)
            min = count;
    }

    // Return minimum count
    return min;
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 2, 3, 4, 3};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Minimum number of multiplications is %d ",
           MatrixChainOrder(arr, 1, n-1));

    getchar();
    return 0;
}
```

Run on IDE

Java

```

/* A naive recursive implementation that simply follows
   the above optimal substructure property */
class MatrixChainMultiplication
{
    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
    static int MatrixChainOrder(int p[], int i, int j)
    {
        if (i == j)
            return 0;

        int min = Integer.MAX_VALUE;

        // place parenthesis at different places between first
        // and last matrix, recursively calculate count of
        // multiplications for each parenthesis placement and
        // return the minimum count
        for (int k=i; k<j; k++)
        {
            int count = MatrixChainOrder(p, i, k) +
                        MatrixChainOrder(p, k+1, j) +
                        p[i-1]*p[k]*p[j];

            if (count < min)
                min = count;
        }

        // Return minimum count
        return min;
    }

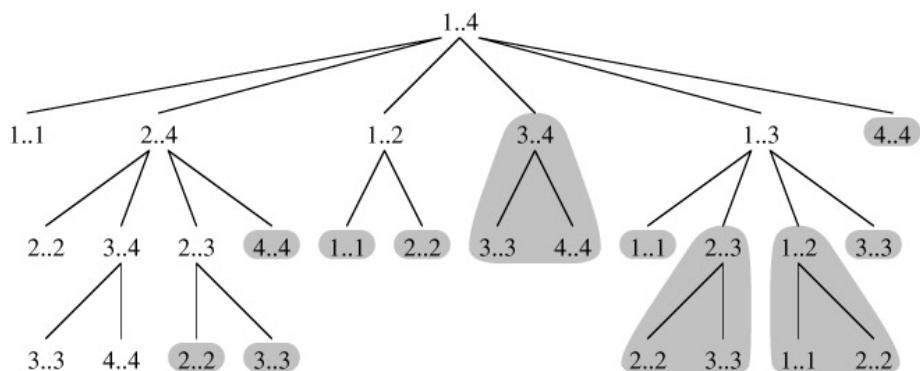
    // Driver program to test above function
    public static void main(String args[])
    {
        int arr[] = new int[] {1, 2, 3, 4, 3};
        int n = arr.length;

        System.out.println("Minimum number of multiplications is "+
                           MatrixChainOrder(arr, 1, n-1));
    }
}
/* This code is contributed by Rajat Mishra*/

```

Run on IDE

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. See the following recursion tree for a matrix chain of size 4. The function `MatrixChainOrder(p, 3, 4)` is called two times. We can see that there are many subproblems being called more than once.



Since same subproblems are called again, this problem has Overlapping Subproblems property. So Matrix Chain Multiplication problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array $m[][]$ in bottom up manner.

Dynamic Programming Solution

Following is C/C++ implementation for Matrix Chain Multiplication problem using Dynamic Programming.

```
// See the Cormen book for details of the following algorithm
#include<stdio.h>
#include<limits.h>

// Matrix A[i] has dimension p[i-1] x p[i] for i = 1..n
int MatrixChainOrder(int p[], int n)
{
    /* For simplicity of the program, one extra row and one
       extra column are allocated in m[][]|. 0th row and 0th
       column of m[][]| are not used */
    int m[n][n];

    int i, j, k, L, q;

    /* m[i,j] = Minimum number of scalar multiplications needed
       to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
       dimension of A[i] is p[i-1] x p[i] */

    // cost is zero when multiplying one matrix.
    for (i=1; i<n; i++)
        m[i][i] = 0;

    // L is chain length.
    for (L=2; L<n; L++)
    {
        for (i=1; i<n-L+1; i++)
        {
            j = i+L-1;
            m[i][j] = INT_MAX;
            for (k=i; k<=j-1; k++)
            {
                // q = cost/scalar multiplications
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    return m[1][n-1];
}

int main()
{
    int arr[] = {1, 2, 3, 4};
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Minimum number of multiplications is %d ",
           MatrixChainOrder(arr, size));

    getchar();
    return 0;
}
```

Run on IDE

Java

```

// Dynamic Programming Python implementation of Matrix
// Chain Multiplication.
// See the Cormen book for details of the following algorithm
class MatrixChainMultiplication
{
    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
    static int MatrixChainOrder(int p[], int n)
    {
        /* For simplicity of the program, one extra row and one
        extra column are allocated in m[][]|. 0th row and 0th
        column of m[][]| are not used */
        int m[][] = new int[n][n];

        int i, j, k, L, q;

        /* m[i,j] = Minimum number of scalar multiplications needed
        to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
        dimension of A[i] is p[i-1] x p[i] */

        // cost is zero when multiplying one matrix.
        for (i = 1; i < n; i++)
            m[i][i] = 0;

        // L is chain length.
        for (L=2; L<n; L++)
        {
            for (i=1; i<n-L+1; i++)
            {
                j = i+L-1;
                if(j == n) continue;
                m[i][j] = Integer.MAX_VALUE;
                for (k=i; k<=j-1; k++)
                {
                    // q = cost/scalar multiplications
                    q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                    if (q < m[i][j])
                        m[i][j] = q;
                }
            }
        }

        return m[1][n-1];
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        int arr[] = new int[] {1, 2, 3, 4};
        int size = arr.length;

        System.out.println("Minimum number of multiplications is "+
                           MatrixChainOrder(arr, size));
    }
}
/* This code is contributed by Rajat Mishra*/

```

Run on IDE

Python

```

# Dynamic Programming Python implementation of Matrix
# Chain Multiplication. See the Cormen book for details
# of the following algorithm
import sys

# Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
def MatrixChainOrder(p, n):

```

```

# For simplicity of the program, one extra row and one
# extra column are allocated in m[][] . 0th row and 0th
# column of m[][] are not used
m = [[0 for x in range(n)] for x in range(n)]

# m[i,j] = Minimum number of scalar multiplications needed
# to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
# dimension of A[i] is p[i-1] x p[i]

# cost is zero when multiplying one matrix.
for i in range(1, n):
    m[i][i] = 0

# L is chain length.
for L in range(2, n):
    for i in range(1, n-L+1):
        j = i+L-1
        m[i][j] = sys.maxint
        for k in range(i, j):

            # q = cost/scalar multiplications
            q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]
            if q < m[i][j]:
                m[i][j] = q

return m[1][n-1]

# Driver program to test above function
arr = [1, 2, 3 ,4]
size = len(arr)

print("Minimum number of multiplications is " +
      str(MatrixChainOrder(arr, size)))
# This Code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

```
Minimum number of multiplications is 18
```

Time Complexity: O(n^3)

Auxiliary Space: O(n^2)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Applications:

Minimum and Maximum values of an expression with * and +

References:

http://en.wikipedia.org/wiki/Matrix_chain_multiplication

<http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Dynamic/chainMatrixMult.htm>



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.1

Average Difficulty : 4.1/5.0
Based on 127 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Dynamic Programming | Set 9 (Binomial Coefficient)

Following are common definition of [Binomial Coefficients](#).

- 1) A [binomial coefficient](#) $C(n, k)$ can be defined as the coefficient of X^k in the expansion of $(1 + X)^n$.
- 2) A binomial coefficient $C(n, k)$ also gives the number of ways, disregarding order, that k objects can be chosen from among n objects; more formally, the number of k -element subsets (or k -combinations) of an n -element set.

The Problem

Write a function that takes two parameters n and k and returns the value of Binomial Coefficient $C(n, k)$. For example, your function should return 6 for $n = 4$ and $k = 2$, and it should return 10 for $n = 5$ and $k = 2$.

1) Optimal Substructure

The value of $C(n, k)$ can be recursively calculated using following standard formula for Binomial Coefficients.

$$\begin{aligned} C(n, k) &= C(n-1, k-1) + C(n-1, k) \\ C(n, 0) &= C(n, n) = 1 \end{aligned}$$

Following is a simple recursive implementation that simply follows the recursive structure mentioned above.

```
// A Naive Recursive Implementation
#include<stdio.h>

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    // Base Cases
    if (k==0 || k==n)
        return 1;

    // Recur
    return binomialCoeff(n-1, k-1) + binomialCoeff(n-1, k);
}

/* Driver program to test above function*/
int main()
{
    int n = 5, k = 2;
    printf("Value of C(%d, %d) is %d ", n, k, binomialCoeff(n, k));
    return 0;
}
```

[Run on IDE](#)

Python

```
# A naive recursive Python implementation

def binomialCoeff(n , k):

    if k==0 or k ==n :
        return 1

    # Recursive Call
    return binomialCoeff(n-1 , k-1) + binomialCoeff(n-1 , k)

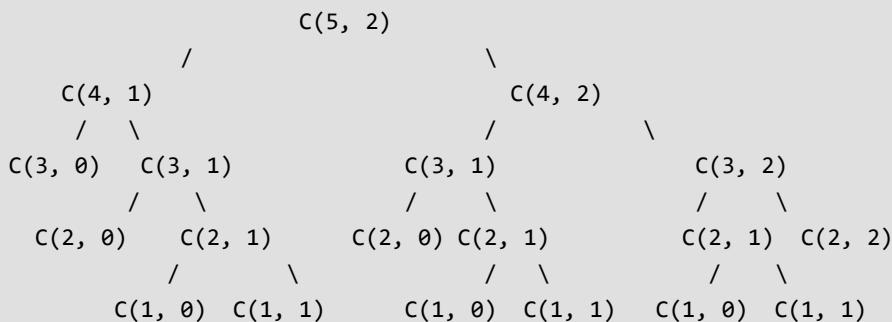
# Driver Program to test ht above function
n = 5
k = 2
print "Value of C(%d,%d) is (%d)" %(n , k , binomialCoeff(n , k))

# This code is contributed by Nikhil Kumar Singh (nickzuck_007)
```

[Run on IDE](#)

2) Overlapping Subproblems

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree for $n = 5$ and $k = 2$. The function $C(3, 1)$ is called two times. For large values of n , there will be many common subproblems.



Since same subproblems are called again, this problem has Overlapping Subproblems property. So the Binomial Coefficient problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, re-computations of same subproblems can be avoided by constructing a temporary array $C[][]$ in bottom up manner. Following is Dynamic Programming based implementation.

```
// A Dynamic Programming based solution that uses table C[][] to
// calculate the Binomial Coefficient
#include<stdio.h>

// Prototype of a utility function that returns minimum of two integers
int min(int a, int b);

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    int C[n+1][k+1];
    int i, j;

    // Calculate value of Binomial Coefficient in bottom up manner
    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= min(i, k); j++)
            if (j == 0 || j == i)
                C[i][j] = 1;
            else
                C[i][j] = (C[i-1][j-1] + C[i-1][j]);
    }
}
```

```

    {
        // Base Cases
        if (j == 0 || j == i)
            C[i][j] = 1;

        // Calculate value using previously stored values
        else
            C[i][j] = C[i-1][j-1] + C[i-1][j];
    }

    return C[n][k];
}

// A utility function to return minimum of two integers
int min(int a, int b)
{
    return (a < b)? a: b;
}

/* Driver program to test above function*/
int main()
{
    int n = 5, k = 2;
    printf ("Value of C(%d, %d) is %d ", n, k, binomialCoeff(n, k) );
    return 0;
}

```

[Run on IDE](#)

Java

```

// A Dynamic Programming based solution that uses table C[][] to
// calculate the Binomial Coefficient

class BinomialCoefficient
{
    // Returns value of Binomial Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int C[][] = new int[n+1][k+1];
        int i, j;

        // Calculate value of Binomial Coefficient in bottom up manner
        for (i = 0; i <= n; i++)
        {
            for (j = 0; j <= min(i, k); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using previously stored values
                else
                    C[i][j] = C[i-1][j-1] + C[i-1][j];
            }
        }

        return C[n][k];
    }

    // A utility function to return minimum of two integers
    static int min(int a, int b)
    {
        return (a < b)? a: b;
    }

    /* Driver program to test above function*/
    public static void main(String args[])
    {
        int n = 5, k = 2;
    }
}

```

```

        System.out.println("Value of C(" + n + "," + k + ") is " + binomialCoeff(n, k));
    }
    /*This code is contributed by Rajat Mishra*/

```

[Run on IDE](#)

Python

```

# A Dynamic Programming based Python Program that uses table C[][][]
# to calculate the Binomial Coefficient

# Returns value of Binomial Coefficient C(n, k)
def binomialCoef(n, k):
    C = [[0 for x in range(k+1)] for x in range(n+1)]

    # Calculate value of Binomial Coefficient in bottom up manner
    for i in range(n+1):
        for j in range(min(i, k)+1):
            # Base Cases
            if j == 0 or j == i:
                C[i][j] = 1

            # Calculate value using previosly stored values
            else:
                C[i][j] = C[i-1][j-1] + C[i-1][j]

    return C[n][k]

# Driver program to test above function
n = 5
k = 2
print("Value of C[" + str(n) + "][" + str(k) + "] is "
      + str(binomialCoef(n,k)))

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

```
Value of C[5][2] is 10
```

Time Complexity: O(n*k)

Auxiliary Space: O(n*k)

Following is a space optimized version of the above code. The following code only uses O(k). Thanks to AK for suggesting this method.

```

// C++ program for space optimized Dynamic Programming
// Solution of Binomial Coefficient
#include<bits/stdc++.h>
using namespace std;

int binomialCoeff(int n, int k)
{
    int C[k+1];
    memset(C, 0, sizeof(C));
    C[0] = 1; // nC0 is 1

```

```

for (int i = 1; i <= n; i++)
{
    // Compute next row of pascal triangle using
    // the previous row
    for (int j = min(i, k); j > 0; j--)
        C[j] = C[j] + C[j-1];
}
return C[k];
}

/* Drier program to test above function*/
int main()
{
    int n = 5, k = 2;
    printf ("Value of C(%d, %d) is %d ",
           n, k, binomialCoeff(n, k));
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program for Optimized Dynamic Programming solution to
# Binomial Coefficient. This one uses the concept of pascal
# Triangle and less memory

def binomialCoeff(n , k):

    # Declaring an empty array
    C = [0 for i in xrange(k+1)]
    C[0] = 1 #since nC0 is 1

    for i in range(1,n+1):

        # Compute next row of pascal triangle using
        # the previous row
        j = min(i ,k)
        while (j>0):
            C[j] = C[j] + C[j-1]
            j -= 1

    return C[k]

# Driver Program to test the above function
n = 5
k = 2
print "Value of C(%d,%d) is %d" %(n,k,binomialCoeff(n,k))

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)

```

[Run on IDE](#)

Output:

```
Value of C[5][2] is 10
```

Time Complexity: O(n*k)

Auxiliary Space: O(k)

Explanation:

1=====> n = 0, C(0,0) = 1

1-1=====> n = 1, C(1,0) = 1, C(1,1) = 1

1-2-1=====>> n = 2, C(2,0) = 1, C(2,1) = 2, C(2,2) = 1

1-3-3-1=====>> n = 3, C(3,0) = 1, C(3,1) = 3, C(3,2) = 3, C(3,3)=1

1-4-6-4-1===>> n = 4, C(4,0) = 1, C(4,1) = 4, C(4,2) = 6, C(4,3)=4, C(4,4)=1

So here every loop on i, builds i'th row of pascal triangle, using (i-1)th row

At any time, every element of array C will have some value (ZERO or more) and in next iteration, value for those elements comes from previous iteration.

In statement,

$C[j] = C[j] + C[j-1]$

Right hand side represents the value coming from previous iteration (A row of Pascal's triangle depends on previous row). Left Hand side represents the value of current iteration which will be obtained by this statement.

Let's say we want to calculate $C(4, 3)$,
i.e. n=4, k=3:

All elements of array C of size 4 (k+1) are initialized to ZERO.

i.e. $C[0] = C[1] = C[2] = C[3] = C[4] = 0$;

Then $C[0]$ is set to 1

For i = 1:

$C[1] = C[1] + C[0] = 0 + 1 = 1 ==>> C(1,1) = 1$

For i = 2:

$C[2] = C[2] + C[1] = 0 + 1 = 1 ==>> C(2,2) = 1$

$C[1] = C[1] + C[0] = 1 + 1 = 2 ==>> C(2,2) = 2$

For i=3:

$C[3] = C[3] + C[2] = 0 + 1 = 1 ==>> C(3,3) = 1$

$C[2] = C[2] + C[1] = 1 + 2 = 3 ==>> C(3,2) = 3$

$C[1] = C[1] + C[0] = 2 + 1 = 3 ==>> C(3,1) = 3$

For i=4:

$C[4] = C[4] + C[3] = 0 + 1 = 1 ==>> C(4,4) = 1$

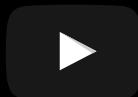
$C[3] = C[3] + C[2] = 1 + 3 = 4 ==>> C(4,3) = 4$

$C[2] = C[2] + C[1] = 3 + 3 = 6 ==>> C(4,2) = 6$

$C[1] = C[1] + C[0] = 3 + 1 = 4 ==>> C(4,1) = 4$

$C(4,3) = 4$ is would be the answer in our example.

See this for [Space and time efficient Binomial Coefficient](#)



References:

<http://www.csl.mtu.edu/cs4321/www/Lectures/Lecture%2015%20-Dynamic%20Programming%20Binomial%20Coefficients.htm>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Mathematical](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.4

Average Difficulty : **2.4/5.0**
Based on **61** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 10 (0-1 Knapsack Problem)

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

We strongly recommend that you click here and practice it, before moving on to the solution.

A simple solution is to consider all subsets of items and calculate the total weight and value of all subsets. Consider the only subsets whose total weight is smaller than W. From all such subsets, pick the maximum value subset.

1) Optimal Substructure:

To consider all subsets of items, there can be two cases for every item: (1) the item is included in the optimal subset, (2) not included in the optimal set.

Therefore, the maximum value that can be obtained from n items is max of following two values.

1) Maximum value obtained by n-1 items and W weight (excluding nth item).

2) Value of nth item plus maximum value obtained by n-1 items and W minus weight of the nth item (including nth item).

If weight of nth item is greater than W, then the nth item cannot be included and case 1 is the only possibility.

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

```
/* A Naive recursive implementation of 0-1 Knapsack problem */
#include<stdio.h>

// A utility function that returns maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
    // Base Case
}
```

```

if (n == 0 || w == 0)
    return 0;

// If weight of the nth item is more than Knapsack capacity W, then
// this item cannot be included in the optimal solution
if (wt[n-1] > w)
    return knapSack(W, wt, val, n-1);

// Return the maximum of two cases:
// (1) nth item included
// (2) not included
else return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                  knapSack(W, wt, val, n-1)
                );
}

// Driver program to test above function
int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}

```

Run on IDE

Java

```

/* A Naive recursive implementation of 0-1 Knapsack problem */
class Knapsack
{

    // A utility function that returns maximum of two integers
    static int max(int a, int b) { return (a > b)? a : b; }

    // Returns the maximum value that can be put in a knapsack of capacity W
    static int knapSack(int W, int wt[], int val[], int n)
    {
        // Base Case
        if (n == 0 || W == 0)
            return 0;

        // If weight of the nth item is more than Knapsack capacity W, then
        // this item cannot be included in the optimal solution
        if (wt[n-1] > W)
            return knapSack(W, wt, val, n-1);

        // Return the maximum of two cases:
        // (1) nth item included
        // (2) not included
        else return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                          knapSack(W, wt, val, n-1)
                        );
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        int val[] = new int[]{60, 100, 120};
        int wt[] = new int[]{10, 20, 30};
        int W = 50;
        int n = val.length;
        System.out.println(knapSack(W, wt, val, n));
    }
}
/*This code is contributed by Rajat Mishra */

```

Run on IDE

Python

```
#A naive recursive implementation of 0-1 Knapsack Problem

# Returns the maximum value that can be put in a knapsack of
# capacity W
def knapSack(W , wt , val , n):

    # Base Case
    if n == 0 or W == 0 :
        return 0

    # If weight of the nth item is more than Knapsack of capacity
    # W, then this item cannot be included in the optimal solution
    if (wt[n-1] > W):
        return knapSack(W , wt , val , n-1)

    # return the maximum of two cases:
    # (1) nth item included
    # (2) not included
    else:
        return max(val[n-1] + knapSack(W-wt[n-1] , wt , val , n-1),
                   knapSack(W , wt , val , n-1))

# end of function knapSack

# To test above function
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print knapSack(W , wt , val , n)

# This code is contributed by Nikhil Kumar Singh
```

[Run on IDE](#)

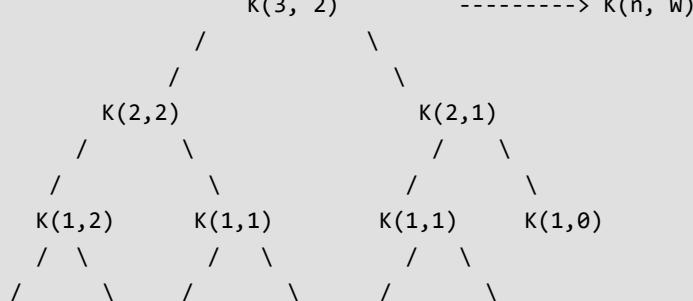
Output:

220

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree, K(1, 1) is being evaluated twice. Time complexity of this naive recursive solution is exponential (2^n).

In the following recursion tree, K() refers to knapSack(). The two parameters indicated in the following recursion tree are n and W.
The recursion tree is for following sample inputs.

`wt[] = {1, 1, 1}, W = 2, val[] = {10, 20, 30}`



```
K(0,2) K(0,1) K(0,1) K(0,0) K(0,1) K(0,0)
Recursion tree for Knapsack capacity 2 units and 3 items of 1 unit weight.
```

Since subproblems are evaluated again, this problem has Overlapping Subproblems property. So the 0-1 Knapsack problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array $K[][]$ in bottom up manner. Following is Dynamic Programming based implementation.

```
// A Dynamic Programming based solution for 0-1 Knapsack problem
#include<stdio.h>

// A utility function that returns maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
    int i, w;
    int K[n+1][W+1];

    // Build table K[][] in bottom up manner
    for (i = 0; i <= n; i++)
    {
        for (w = 0; w <= W; w++)
        {
            if (i==0 || w==0)
                K[i][w] = 0;
            else if (wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
            else
                K[i][w] = K[i-1][w];
        }
    }

    return K[n][W];
}

int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

[Run on IDE](#)

Java

```
// A Dynamic Programming based solution for 0-1 Knapsack problem
class Knapsack
{

    // A utility function that returns maximum of two integers
    static int max(int a, int b) { return (a > b)? a : b; }

    // Returns the maximum value that can be put in a knapsack of capacity W
    static int knapSack(int W, int wt[], int val[], int n)
    {
        int i, w;
        int K[][] = new int[n+1][W+1];
```

```
// Build table K[][] in bottom up manner
for (i = 0; i <= n; i++)
{
    for (w = 0; w <= W; w++)
    {
        if (i==0 || w==0)
            K[i][w] = 0;
        else if (wt[i-1] <= w)
            K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
        else
            K[i][w] = K[i-1][w];
    }
}

return K[n][W];
}

// Driver program to test above function
public static void main(String args[])
{
    int val[] = new int[]{60, 100, 120};
    int wt[] = new int[]{10, 20, 30};
    int W = 50;
    int n = val.length;
    System.out.println(knapSack(W, wt, val, n));
}
/*This code is contributed by Rajat Mishra */
```

[Run on IDE](#)

Python

```
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]

# Driver program to test above function
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(knapSack(W, wt, val, n))

# This code is contributed by Bhavya Jain
```

[Run on IDE](#)

Output:

220

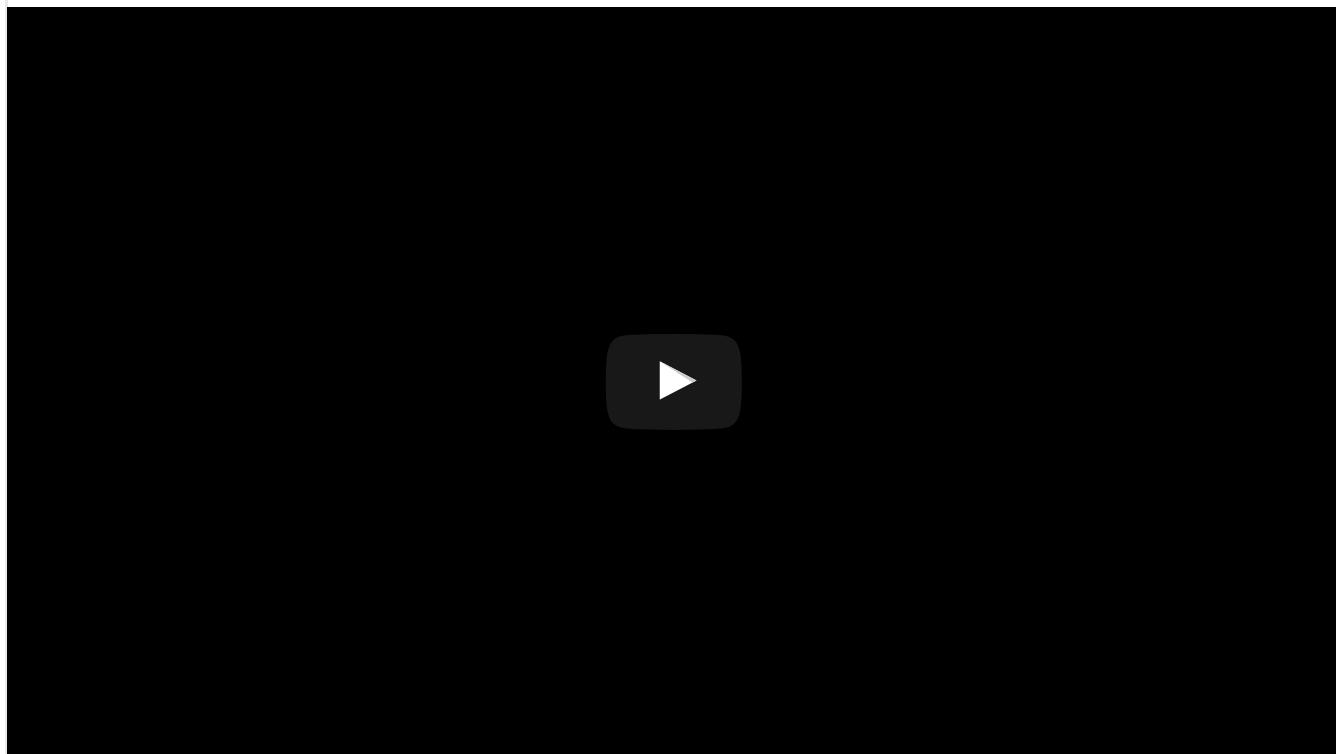
Time Complexity: $O(nW)$ where n is the number of items and W is the capacity of knapsack.

Asked in: Amazon

References:

<http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf>

<http://www.cse.unl.edu/~goddard/Courses/CSCE310J/Lectures/Lecture8-DynamicProgramming.pdf>



Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Dynamic Programming

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **126** vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

Practice

GATE CS

Placements

GeeksQuiz

Google™ Custom Search



Login/Register

Dynamic Programming | Set 11 (Egg Dropping Puzzle)

The following is a description of the instance of this famous puzzle involving $n=2$ eggs and a building with $k=36$ floors.

Suppose that we wish to know which stories in a 36-story building are safe to drop eggs from, and which will cause the eggs to break on landing. We make a few assumptions:

.....An egg that survives a fall can be used again.

.....A broken egg must be discarded.

.....The effect of a fall is the same for all eggs.

.....If an egg breaks when dropped, then it would break if dropped from a higher floor.

.....If an egg survives a fall then it would survive a shorter fall.

.....It is not ruled out that the first-floor windows break eggs, nor is it ruled out that the 36th-floor do not cause an egg to break.

If only one egg is available and we wish to be sure of obtaining the right result, the experiment can be carried out in only one way. Drop the egg from the first-floor window; if it survives, drop it from the second floor window. Continue upward until it breaks. In the worst case, this method may require 36 droppings. Suppose 2 eggs are available. What is the least number of egg-droppings that is guaranteed to work in all cases?

The problem is not actually to find the critical floor, but merely to decide floors from which eggs should be dropped so that total number of trials are minimized.

Source: [Wiki for Dynamic Programming](#)

We strongly recommend that you click here and practice it, before moving on to the solution.

In this post, we will discuss solution to a general problem with n eggs and k floors. The solution is to try dropping an egg from every floor (from 1 to k) and recursively calculate the minimum number of droppings needed in worst case. The floor which gives the minimum value in worst case is going to be part of the solution.

In the following solutions, we return the minimum number of trials in worst case; these solutions can be easily modified to print floor numbers of every trials also.

1) Optimal Substructure:

When we drop an egg from a floor x , there can be two cases (1) The egg breaks (2) The egg doesn't break.

1) If the egg breaks after dropping from x th floor, then we only need to check for floors lower than x with remaining eggs; so the problem reduces to $x-1$ floors and $n-1$ eggs

2) If the egg doesn't break after dropping from the x th floor, then we only need to check for floors higher than x ; so the problem reduces to $k-x$ floors and n eggs.

Since we need to minimize the number of trials in *worst case*, we take the maximum of two cases. We consider the max of above two cases for every floor and choose the floor which yields minimum number of trials.

```

k ==> Number of floors
n ==> Number of Eggs
eggDrop(n, k) ==> Minimum number of trials needed to find the critical
                     floor in worst case.
eggDrop(n, k) = 1 + min{max(eggDrop(n - 1, x - 1), eggDrop(n, k - x)):
                     x in {1, 2, ..., k}}

```

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

```

#include <stdio.h>
#include <limits.h>

// A utility function to get maximum of two integers
int max(int a, int b) { return (a > b)? a: b; }

/* Function to get minimum number of trials needed in worst
   case with n eggs and k floors */
int eggDrop(int n, int k)
{
    // If there are no floors, then no trials needed. OR if there is
    // one floor, one trial needed.
    if (k == 1 || k == 0)
        return k;

    // We need k trials for one egg and k floors
    if (n == 1)
        return k;

    int min = INT_MAX, x, res;

    // Consider all droppings from 1st floor to kth floor and
    // return the minimum of these values plus 1.
    for (x = 1; x <= k; x++)
    {
        res = max(eggDrop(n-1, x-1), eggDrop(n, k-x));
        if (res < min)
            min = res;
    }

    return min + 1;
}

/* Driver program to test to printDups*/
int main()
{
    int n = 2, k = 10;
    printf ("\nMinimum number of trials in worst case with %d eggs and "
           "%d floors is %d \n", n, k, eggDrop(n, k));
    return 0;
}

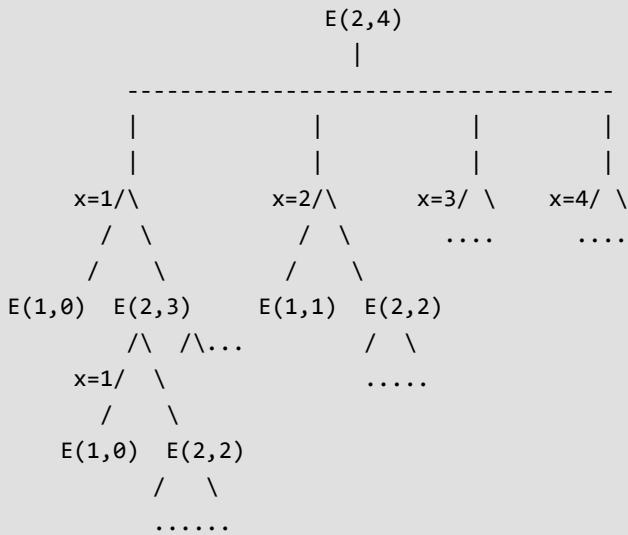
```

Run on IDE

Output:

```
Minimum number of trials in worst case with 2 eggs and 10 floors is 4
```

It should be noted that the above function computes the same subproblems again and again. See the following partial recursion tree, $E(2, 2)$ is being evaluated twice. There will many repeated subproblems when you draw the complete recursion tree even for small values of n and k .



Partial recursion tree for 2 eggs and 4 floors.

Since same subproblems are called again, this problem has Overlapping Subproblems property. So Egg Dropping Puzzle has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array eggFloor[][] in bottom up manner.

Dynamic Programming Solution

Following are C++ and Python implementations for Egg Dropping problem using Dynamic Programming.

```
# A Dynamic Programming based C++ Program for the Egg Dropping Puzzle
# include <stdio.h>
# include <limits.h>

// A utility function to get maximum of two integers
int max(int a, int b) { return (a > b)? a: b; }

/* Function to get minimum number of trials needed in worst
   case with n eggs and k floors */
int eggDrop(int n, int k)
{
    /* A 2D table where entry eggFloor[i][j] will represent minimum
       number of trials needed for i eggs and j floors. */
    int eggFloor[n+1][k+1];
    int res;
    int i, j, x;

    // We need one trial for one floor and 0 trials for 0 floors
    for (i = 1; i <= n; i++)
    {
        eggFloor[i][1] = 1;
        eggFloor[i][0] = 0;
    }
}
```

```

// We always need j trials for one egg and j floors.
for (j = 1; j <= k; j++)
    eggFloor[1][j] = j;

// Fill rest of the entries in table using optimal substructure
// property
for (i = 2; i <= n; i++)
{
    for (j = 2; j <= k; j++)
    {
        eggFloor[i][j] = INT_MAX;
        for (x = 1; x <= j; x++)
        {
            res = 1 + max(eggFloor[i-1][x-1], eggFloor[i][j-x]);
            if (res < eggFloor[i][j])
                eggFloor[i][j] = res;
        }
    }
}

// eggFloor[n][k] holds the result
return eggFloor[n][k];
}

/* Driver program to test to printDups*/
int main()
{
    int n = 2, k = 36;
    printf ("\nMinimum number of trials in worst case with %d eggs and "
           "%d floors is %d \n", n, k, eggDrop(n, k));
    return 0;
}

```

[Run on IDE](#)

Java

```

//A Dynamic Programming based Python Program for the Egg Dropping Puzzle
class EggDrop
{
    // A utility function to get maximum of two integers
    static int max(int a, int b) { return (a > b)? a: b; }

    /* Function to get minimum number of trials needed in worst
    case with n eggs and k floors */
    static int eggDrop(int n, int k)
    {
        /* A 2D table where entry eggFloor[i][j] will represent minimum
        number of trials needed for i eggs and j floors. */
        int eggFloor[][] = new int[n+1][k+1];
        int res;
        int i, j, x;

        // We need one trial for one floor and 0 trials for 0 floors
        for (i = 1; i <= n; i++)
        {
            eggFloor[i][1] = 1;
            eggFloor[i][0] = 0;
        }

        // We always need j trials for one egg and j floors.
        for (j = 1; j <= k; j++)
            eggFloor[1][j] = j;

        // Fill rest of the entries in table using optimal substructure
        // property
        for (i = 2; i <= n; i++)
        {
            for (j = 2; j <= k; j++)
            {

```

```

        eggFloor[i][j] = Integer.MAX_VALUE;
        for (x = 1; x <= j; x++)
        {
            res = 1 + max(eggFloor[i-1][x-1], eggFloor[i][j-x]);
            if (res < eggFloor[i][j])
                eggFloor[i][j] = res;
        }
    }

    // eggFloor[n][k] holds the result
    return eggFloor[n][k];
}

/* Driver program to test to printDups*/
public static void main(String args[])
{
    int n = 2, k = 10;
    System.out.println("Minimum number of trials in worst case with "+n+" eggs and "+k+
                       " floors is "+eggDrop(n, k));
}
/*This code is contributed by Rajat Mishra*/

```

[Run on IDE](#)

Python

```

# A Dynamic Programming based Python Program for the Egg Dropping Puzzle
INT_MAX = 32767

# Function to get minimum number of trials needed in worst
# case with n eggs and k floors
def eggDrop(n, k):
    # A 2D table where entry eggFloor[i][j] will represent minimum
    # number of trials needed for i eggs and j floors.
    eggFloor = [[0 for x in range(k+1)] for x in range(n+1)]

    # We need one trial for one floor and 0 trials for 0 floors
    for i in range(1, n+1):
        eggFloor[i][1] = 1
        eggFloor[i][0] = 0

    # We always need j trials for one egg and j floors.
    for j in range(1, k+1):
        eggFloor[1][j] = j

    # Fill rest of the entries in table using optimal substructure
    # property
    for i in range(2, n+1):
        for j in range(2, k+1):
            eggFloor[i][j] = INT_MAX
            for x in range(1, j+1):
                res = 1 + max(eggFloor[i-1][x-1], eggFloor[i][j-x])
                if res < eggFloor[i][j]:
                    eggFloor[i][j] = res

    # eggFloor[n][k] holds the result
    return eggFloor[n][k]

# Driver program to test to printDups
n = 2
k = 36
print("Minimum number of trials in worst case with " + str(n) + " eggs and " +
      + str(k) + " floors is " + str(eggDrop(n, k)))

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

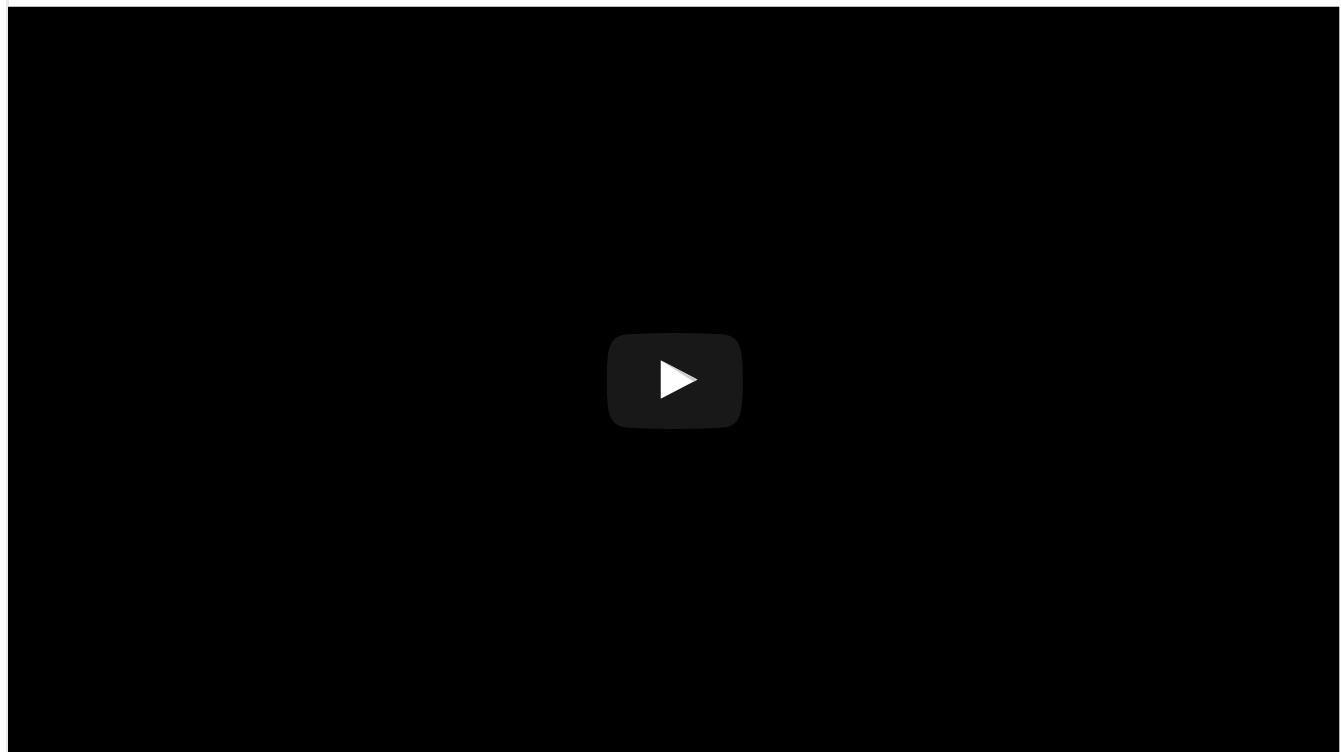
Minimum number of trials in worst case with 2 eggs and 36 floors is 8

Time Complexity: $O(nk^2)$

Auxiliary Space: $O(nk)$

As an exercise, you may try modifying the above DP solution to print all intermediate floors (The floors used for minimum trial solution).

2 Eggs and 100 Floor Puzzle



References:

<http://archive.ite.journal.informs.org/Vol4No1/Sniedovich/index.php>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming

Dynamic Programming

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

4.3

Average Difficulty : 4.3/5.0
Based on 94 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 12 (Longest Palindromic Subsequence)

Given a sequence, find the length of the longest palindromic subsequence in it. For example, if the given sequence is "BBABCBCAB", then the output should be 7 as "BABCBAB" is the longest palindromic subsequence in it. "BBBBB" and "BBCBB" are also palindromic subsequences of the given sequence, but not the longest ones.

The naive solution for this problem is to generate all subsequences of the given sequence and find the longest palindromic subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

1) Optimal Substructure:

Let $X[0..n-1]$ be the input sequence of length n and $L(0, n-1)$ be the length of the longest palindromic subsequence of $X[0..n-1]$.

If last and first characters of X are same, then $L(0, n-1) = L(1, n-2) + 2$.

Else $L(0, n-1) = \text{MAX } (L(1, n-1), L(0, n-2))$.

Following is a general recursive solution with all cases handled.

```
// Every single character is a palindrome of length 1
L(i, i) = 1 for all indexes i in given sequence

// IF first and last characters are not same
If (X[i] != X[j]) L(i, j) = max{L(i + 1, j), L(i, j - 1)}

// If there are only 2 characters and both are same
Else if (j == i + 1) L(i, j) = 2

// If there are more than two characters, and first and last
// characters are same
Else L(i, j) = L(i + 1, j - 1) + 2
```

2) Overlapping Subproblems

Following is simple recursive implementation of the LPS problem. The implementation simply follows the recursive structure mentioned above.

```
#include<stdio.h>
#include<string.h>

// A utility function to get max of two integers
int max (int x, int y) { return (x > y)? x : y; }
```

```

// Returns the length of the longest palindromic subsequence in seq
int lps(char *seq, int i, int j)
{
    // Base Case 1: If there is only 1 character
    if (i == j)
        return 1;

    // Base Case 2: If there are only 2 characters and both are same
    if (seq[i] == seq[j] && i + 1 == j)
        return 2;

    // If the first and last characters match
    if (seq[i] == seq[j])
        return lps (seq, i+1, j-1) + 2;

    // If the first and last characters do not match
    return max( lps(seq, i, j-1), lps(seq, i+1, j) );
}

/* Driver program to test above functions */
int main()
{
    char seq[] = "GEEKSFORGEEKS";
    int n = strlen(seq);
    printf ("The length of the LPS is %d", lps(seq, 0, n-1));
    getchar();
    return 0;
}

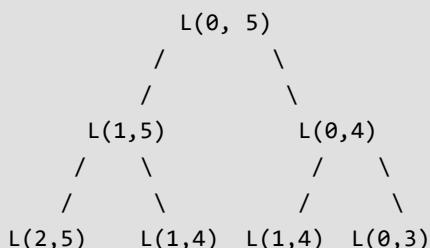
```

Run on IDE

Output:

The length of the LPS is 5

Considering the above implementation, following is a partial recursion tree for a sequence of length 6 with all different characters.



In the above partial recursion tree, L(1, 4) is being solved twice. If we draw the complete recursion tree, then we can see that there are many subproblems which are solved again and again. Since same subproblems are called again, this problem has Overlapping Subproblems property. So LPS problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array L[][] in bottom up manner.

Dynamic Programming Solution

```

# A Dynamic Programming based Python program for LPS problem
# Returns the length of the longest palindromic subsequence in seq
#include<stdio.h>
#include<string.h>

```

```

// A utility function to get max of two integers
int max (int x, int y) { return (x > y)? x : y; }

// Returns the length of the longest palindromic subsequence in seq
int lps(char *str)
{
    int n = strlen(str);
    int i, j, cl;
    int L[n][n]; // Create a table to store results of subproblems

    // Strings of length 1 are palindrome of length 1
    for (i = 0; i < n; i++)
        L[i][i] = 1;

    // Build the table. Note that the lower diagonal values of table are
    // useless and not filled in the process. The values are filled in a
    // manner similar to Matrix Chain Multiplication DP solution (See
    // http://www.geeksforgeeks.org/archives/15553). cl is length of
    // substring
    for (cl=2; cl<=n; cl++)
    {
        for (i=0; i<n-cl+1; i++)
        {
            j = i+cl-1;
            if (str[i] == str[j] && cl == 2)
                L[i][j] = 2;
            else if (str[i] == str[j])
                L[i][j] = L[i+1][j-1] + 2;
            else
                L[i][j] = max(L[i][j-1], L[i+1][j]);
        }
    }

    return L[0][n-1];
}

/* Driver program to test above functions */
int main()
{
    char seq[] = "GEEKS FOR GEEKS";
    int n = strlen(seq);
    printf ("The length of the LPS is %d", lps(seq));
    getchar();
    return 0;
}

```

[Run on IDE](#)

Java

```

//A Dynamic Programming based Python Program for the Egg Dropping Puzzle
class LPS
{

    // A utility function to get max of two integers
    static int max (int x, int y) { return (x > y)? x : y; }

    // Returns the length of the longest palindromic subsequence in seq
    static int lps(String seq)
    {
        int n = seq.length();
        int i, j, cl;
        int L[][] = new int[n][n]; // Create a table to store results of subproblems

        // Strings of length 1 are palindrome of length 1
        for (i = 0; i < n; i++)
            L[i][i] = 1;

        // Build the table. Note that the lower diagonal values of table are
        // useless and not filled in the process. The values are filled in a

```

```

// manner similar to Matrix Chain Multiplication DP solution (See
// http://www.geeksforgeeks.org/archives/15553). cl is length of
// substring
for (cl=2; cl<=n; cl++)
{
    for (i=0; i<n-cl+1; i++)
    {
        j = i+cl-1;
        if (seq.charAt(i) == seq.charAt(j) && cl == 2)
            L[i][j] = 2;
        else if (seq.charAt(i) == seq.charAt(j))
            L[i][j] = L[i+1][j-1] + 2;
        else
            L[i][j] = max(L[i][j-1], L[i+1][j]);
    }
}

return L[0][n-1];
}

/* Driver program to test above functions */
public static void main(String args[])
{
    String seq = "GEEKSFORGEEKS";
    int n = seq.length();
    System.out.println("The length of the lps is "+ lps(seq));
}
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# A Dynamic Programming based Python program for LPS problem
# Returns the length of the longest palindromic subsequence in seq
def lps(str):
    n = len(str)

    # Create a table to store results of subproblems
    L = [[0 for x in range(n)] for x in range(n)]

    # Strings of length 1 are palindrome of length 1
    for i in range(n):
        L[i][i] = 1

    # Build the table. Note that the lower diagonal values of table are
    # useless and not filled in the process. The values are filled in a
    # manner similar to Matrix Chain Multiplication DP solution (See
    # http://www.geeksforgeeks.org/dynamic-programming-set-8-matrix-chain-multiplication/
    # cl is length of substring
    for cl in range(2, n+1):
        for i in range(n-cl+1):
            j = i+cl-1
            if str[i] == str[j] and cl == 2:
                L[i][j] = 2
            elif str[i] == str[j]:
                L[i][j] = L[i+1][j-1] + 2
            else:
                L[i][j] = max(L[i][j-1], L[i+1][j]);

    return L[0][n-1]

# Driver program to test above functions
seq = "GEEKS FOR GEEKS"
n = len(seq)
print("The length of the LPS is " + str(lps(seq)))

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

```
The length of the LPS is 7
```

Time Complexity of the above implementation is $O(n^2)$ which is much better than the worst case time complexity of Naive Recursive implementation.

This problem is close to the [Longest Common Subsequence \(LCS\) problem](#). In fact, we can use LCS as a subroutine to solve this problem. Following is the two step solution that uses LCS.

- 1) Reverse the given sequence and store the reverse in another array say rev[0..n-1]
- 2) LCS of the given sequence and rev[] will be the longest palindromic sequence.

This solution is also a $O(n^2)$ solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

<http://users.eecs.northwestern.edu/~dda902/336/hw6-sol.pdf>



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [palindrome](#) [subsequence](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence

- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 93 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 13 (Cutting a Rod)

Given a rod of length n inches and an array of prices that contains prices of all pieces of size smaller than n . Determine the maximum value obtainable by cutting up the rod and selling the pieces. For example, if length of the rod is 8 and the values of different pieces are given as following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)

length		1	2	3	4	5	6	7	8
<hr/>									
price		1	5	8	9	10	17	17	20

And if the prices are as following, then the maximum obtainable value is 24 (by cutting in eight pieces of length 1)

length		1	2	3	4	5	6	7	8
<hr/>									
price		3	5	8	9	10	17	17	20

The naive solution for this problem is to generate all configurations of different pieces and find the highest priced configuration. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

1) Optimal Substructure:

We can get the best price by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut.

Let $\text{cutRod}(n)$ be the required (best possible price) value for a rod of length n . $\text{cutRod}(n)$ can be written as following.

$$\text{cutRod}(n) = \max(\text{price}[i] + \text{cutRod}(n-i)) \text{ for all } i \text{ in } \{0, 1 \dots n-1\}$$

2) Overlapping Subproblems

Following is simple recursive implementation of the Rod Cutting problem. The implementation simply follows the recursive structure mentioned above.

```
// A Naive recursive solution for Rod cutting problem
#include<stdio.h>
#include<limits.h>
```

```
// A utility function to get the maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

/* Returns the best obtainable price for a rod of length n and
   price[] as prices of different pieces */
int cutRod(int price[], int n)
{
    if (n <= 0)
        return 0;
    int max_val = INT_MIN;

    // Recursively cut the rod in different pieces and compare different
    // configurations
    for (int i = 0; i<n; i++)
        max_val = max(max_val, price[i] + cutRod(price, n-i-1));

    return max_val;
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Maximum Obtainable Value is %d\n", cutRod(arr, size));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Java

```
// // A Naive recursive solution for Rod cutting problem
class RodCutting
{
    /* Returns the best obtainable price for a rod of length
       n and price[] as prices of different pieces */
    static int cutRod(int price[], int n)
    {
        if (n <= 0)
            return 0;
        int max_val = Integer.MIN_VALUE;

        // Recursively cut the rod in different pieces and
        // compare different configurations
        for (int i = 0; i<n; i++)
            max_val = Math.max(max_val,
                               price[i] + cutRod(price, n-i-1));

        return max_val;
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        int arr[] = new int[] {1, 5, 8, 9, 10, 17, 17, 20};
        int size = arr.length;
        System.out.println("Maximum Obtainable Value is "+
                           cutRod(arr, size));

    }
}
/* This code is contributed by Rajat Mishra */
```

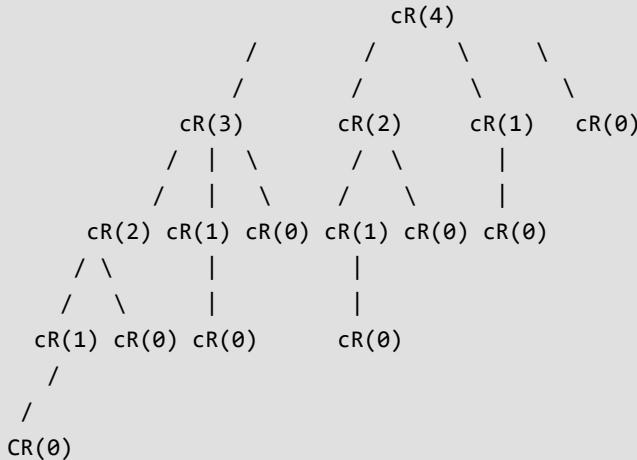
[Run on IDE](#)

Output:

```
Maximum Obtainable Value is 22
```

Considering the above implementation, following is recursion tree for a Rod of length 4.

```
cR() ---> cutRod()
```



In the above partial recursion tree, $cR(2)$ is being solved twice. We can see that there are many subproblems which are solved again and again. Since same subproblems are called again, this problem has Overlapping Subproblems property. So the Rod Cutting problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array $val[]$ in bottom up manner.

```
// A Dynamic Programming solution for Rod cutting problem
#include<stdio.h>
#include<limits.h>

// A utility function to get the maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

/* Returns the best obtainable price for a rod of length n and
   price[] as prices of different pieces */
int cutRod(int price[], int n)
{
    int val[n+1];
    val[0] = 0;
    int i, j;

    // Build the table val[] in bottom up manner and return the last entry
    // from the table
    for (i = 1; i<=n; i++)
    {
        int max_val = INT_MIN;
        for (j = 0; j < i; j++)
            max_val = max(max_val, price[j] + val[i-j-1]);
        val[i] = max_val;
    }

    return val[n];
}

/* Driver program to test above functions */
int main()
{
```

```

int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};
int size = sizeof(arr)/sizeof(arr[0]);
printf("Maximum Obtainable Value is %d\n", cutRod(arr, size));
getchar();
return 0;
}

```

[Run on IDE](#)

Java

```

// A Dynamic Programming solution for Rod cutting problem
class RodCutting
{
    /* Returns the best obtainable price for a rod of
       length n and price[] as prices of different pieces */
    static int cutRod(int price[],int n)
    {
        int val[] = new int[n+1];
        val[0] = 0;

        // Build the table val[] in bottom up manner and return
        // the last entry from the table
        for (int i = 1; i<=n; i++)
        {
            int max_val = Integer.MIN_VALUE;
            for (int j = 0; j < i; j++)
                max_val = Math.max(max_val,
                                   price[j] + val[i-j-1]);
            val[i] = max_val;
        }

        return val[n];
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        int arr[] = new int[] {1, 5, 8, 9, 10, 17, 17, 20};
        int size = arr.length;
        System.out.println("Maximum Obtainable Value is " +
                           cutRod(arr, size));
    }
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# A Dynamic Programming solution for Rod cutting problem
INT_MIN = -32767

# Returns the best obtainable price for a rod of length n and
# price[] as prices of different pieces
def cutRod(price, n):
    val = [0 for x in range(n+1)]
    val[0] = 0

    # Build the table val[] in bottom up manner and return
    # the last entry from the table
    for i in range(1, n+1):
        max_val = INT_MIN
        for j in range(i):
            max_val = max(max_val, price[j] + val[i-j-1])
        val[i] = max_val

    return val[n]

```

```
# Driver program to test above functions
arr = [1, 5, 8, 9, 10, 17, 17, 20]
size = len(arr)
print("Maximum Obtainable Value is " + str(cutRod(arr, size)))

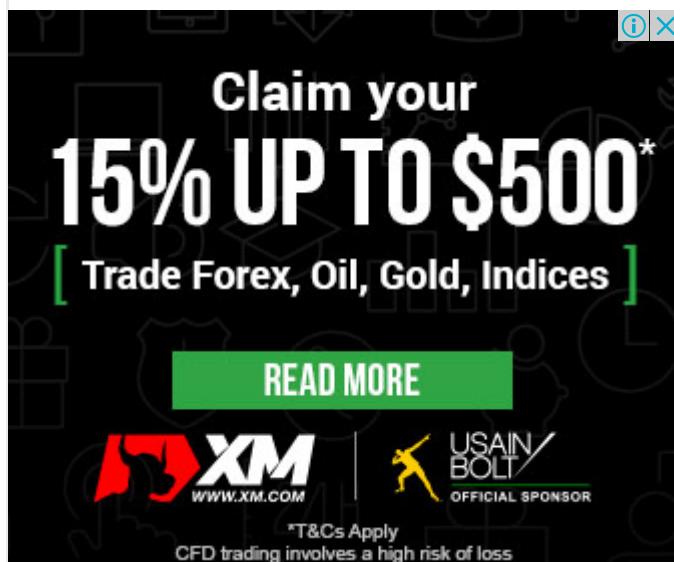
# This code is contributed by Bhavya Jain
```

[Run on IDE](#)**Output:**

```
Maximum Obtainable Value is 22
```

Time Complexity of the above implementation is $O(n^2)$ which is much better than the worst case time complexity of Naive Recursive implementation.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#)[Dynamic Programming](#)

Related Posts:

- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence

(Login to Rate and Mark)

3

Average Difficulty : 3/5.0
Based on 78 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 14 (Maximum Sum Increasing Subsequence)

Given an array of n positive integers. Write a program to find the sum of maximum sum subsequence of the given array such that the integers in the subsequence are sorted in increasing order. For example, if input is {1, 101, 2, 3, 100, 4, 5}, then output should be 106 (1 + 2 + 3 + 100), if the input array is {3, 4, 5, 10}, then output should be 22 (3 + 4 + 5 + 10) and if the input array is {10, 5, 4, 3}, then output should be 10

We strongly recommend that you click here and practice it, before moving on to the solution.

Solution

This problem is a variation of standard [Longest Increasing Subsequence \(LIS\) problem](#). We need a slight change in the Dynamic Programming solution of [LIS problem](#). All we need to change is to use sum as a criteria instead of length of increasing subsequence.

Following are C/C++ and Python implementations for Dynamic Programming solution of the problem.

```
/* Dynamic Programming implementation of Maximum Sum Increasing
Subsequence (MSIS) problem */
#include<stdio.h>

/* maxSumIS() returns the maximum sum of increasing subsequence
   in arr[] of size n */
int maxSumIS( int arr[], int n )
{
    int i, j, max = 0;
    int msis[n];

    /* Initialize msis values for all indexes */
    for ( i = 0; i < n; i++ )
        msis[i] = arr[i];

    /* Compute maximum sum values in bottom up manner */
    for ( i = 1; i < n; i++ )
        for ( j = 0; j < i; j++ )
            if ( arr[i] > arr[j] && msis[i] < msis[j] + arr[i] )
                msis[i] = msis[j] + arr[i];

    /* Pick maximum of all msis values */
    for ( i = 0; i < n; i++ )
        if ( max < msis[i] )
            max = msis[i];
}

int main()
{
    int arr[] = { 1, 101, 2, 3, 100, 4, 5 };
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Maximum Sum Increasing Subsequence is %d", maxSumIS(arr, n));
    return 0;
}
```

```

        max = msis[i];

    return max;
}

/* Driver program to test above function */
int main()
{
    int arr[] = {1, 101, 2, 3, 100, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Sum of maximum sum increasing subsequence is %d\n",
           maxSumIS( arr, n ) );
    return 0;
}

```

[Run on IDE](#)

Java

```

/* Dynamic Programming Java implementation of Maximum Sum
   Increasing Subsequence (MSIS) problem */
class MSIS
{
    /* maxSumIS() returns the maximum sum of increasing
       subsequence in arr[] of size n */
    static int maxSumIS( int arr[], int n )
    {
        int i, j, max = 0;
        int msis[] = new int[n];

        /* Initialize msis values for all indexes */
        for ( i = 0; i < n; i++ )
            msis[i] = arr[i];

        /* Compute maximum sum values in bottom up manner */
        for ( i = 1; i < n; i++ )
            for ( j = 0; j < i; j++ )
                if ( arr[i] > arr[j] &&
                     msis[i] < msis[j] + arr[i])
                    msis[i] = msis[j] + arr[i];

        /* Pick maximum of all msis values */
        for ( i = 0; i < n; i++ )
            if ( max < msis[i] )
                max = msis[i];

        return max;
    }

    /* Driver program to test above function */
    public static void main(String args[])
    {
        int arr[] = new int[]{1, 101, 2, 3, 100, 4, 5};
        int n = arr.length;
        System.out.println("Sum of maximum sum increasing "+
                           "subsequence is "+
                           maxSumIS( arr, n ) );
    }
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```
# Dynamic Programming based Python implementation of Maximum Sum Increasing
# Subsequence (MSIS) problem
```

```
# maxSumIS() returns the maximum sum of increasing subsequence in arr[] of
# size n
def maxSumIS(arr, n):
    max = 0
    msis = [0 for x in range(n)]

    # Initialize msis values for all indexes
    for i in range(n):
        msis[i] = arr[i]

    # Compute maximum sum values in bottom up manner
    for i in range(1, n):
        for j in range(i):
            if arr[i] > arr[j] and msis[i] < msis[j] + arr[i]:
                msis[i] = msis[j] + arr[i]

    # Pick maximum of all msis values
    for i in range(n):
        if max < msis[i]:
            max = msis[i]

    return max

# Driver program to test above function
arr = [1, 101, 2, 3, 100, 4, 5]
n = len(arr)
print("Sum of maximum sum increasing subsequence is " +
      str(maxSumIS(arr, n)))

# This code is contributed by Bhavya Jain
```

[Run on IDE](#)

Output:

Sum of maximum sum increasing subsequence is 106

Time Complexity: $O(n^2)$

Source: Maximum Sum Increasing Subsequence Problem

Asked in: Amazon, Morgan Stanley

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Dynamic Programming LIS

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

2.7

Average Difficulty : 2.7/5.0
Based on 85 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 15 (Longest Bitonic Subsequence)

Given an array $\text{arr}[0 \dots n-1]$ containing n positive integers, a **subsequence** of $\text{arr}[]$ is called Bitonic if it is first increasing, then decreasing. Write a function that takes an array as argument and returns the length of the longest bitonic subsequence.

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, 11, 2, 10, 4, 5, 2, 1};
Output: 6 (A Longest Bitonic Subsequence of length 6 is 1, 2, 10, 4, 2, 1)
```

```
Input arr[] = {12, 11, 40, 5, 3, 1}
Output: 5 (A Longest Bitonic Subsequence of length 5 is 12, 11, 5, 3, 1)
```

```
Input arr[] = {80, 60, 30, 40, 20, 10}
Output: 5 (A Longest Bitonic Subsequence of length 5 is 80, 60, 30, 20, 10)
```

Source: Microsoft Interview Question

Solution

This problem is a variation of standard **Longest Increasing Subsequence (LIS) problem**. Let the input array be $\text{arr}[]$ of length n . We need to construct two arrays $\text{lis}[]$ and $\text{lds}[]$ using Dynamic Programming solution of **LIS problem**. $\text{lis}[i]$ stores the length of the Longest Increasing subsequence ending with $\text{arr}[i]$. $\text{lds}[i]$ stores the length of the longest Decreasing subsequence starting from $\text{arr}[i]$. Finally, we need to return the max value of $\text{lis}[i] + \text{lds}[i] - 1$ where i is from 0 to $n-1$.

Following is C++ implementation of the above Dynamic Programming solution.

```
/* Dynamic Programming implementation of longest bitonic subsequence problem */
#include<stdio.h>
#include<stdlib.h>

/* lbs() returns the length of the Longest Bitonic Subsequence in
   arr[] of size n. The function mainly creates two temporary arrays
   lis[] and lds[] and returns the maximum lis[i] + lds[i] - 1.

   lis[i] ==> Longest Increasing subsequence ending with arr[i]
   lds[i] ==> Longest decreasing subsequence starting with arr[i]
```

```

/*
int lbs( int arr[], int n )
{
    int i, j;

    /* Allocate memory for LIS[] and initialize LIS values as 1 for
       all indexes */
    int *lis = new int[n];
    for (i = 0; i < n; i++)
        lis[i] = 1;

    /* Compute LIS values from left to right */
    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
            if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
                lis[i] = lis[j] + 1;

    /* Allocate memory for lds and initialize LDS values for
       all indexes */
    int *lds = new int [n];
    for (i = 0; i < n; i++)
        lds[i] = 1;

    /* Compute LDS values from right to left */
    for (i = n-2; i >= 0; i--)
        for (j = n-1; j > i; j--)
            if (arr[i] > arr[j] && lds[i] < lds[j] + 1)
                lds[i] = lds[j] + 1;

    /* Return the maximum value of lis[i] + lds[i] - 1*/
    int max = lis[0] + lds[0] - 1;
    for (i = 1; i < n; i++)
        if (lis[i] + lds[i] - 1 > max)
            max = lis[i] + lds[i] - 1;
    return max;
}

/* Driver program to test above function */
int main()
{
    int arr[] = {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5,
                13, 3, 11, 7, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Length of LBS is %d\n", lbs( arr, n ) );
    return 0;
}

```

Run on IDE

Java

```

/* Dynamic Programming implementation in Java for longest bitonic
   subsequence problem */
import java.util.*;
import java.lang.*;
import java.io.*;

class LBS
{
    /* lbs() returns the length of the Longest Bitonic Subsequence in
       arr[] of size n. The function mainly creates two temporary arrays
       lis[] and lds[] and returns the maximum lis[i] + lds[i] - 1.

       lis[i] ==> Longest Increasing subsequence ending with arr[i]
       lds[i] ==> Longest decreasing subsequence starting with arr[i]
    */
    static int lbs( int arr[], int n )
    {
        int i, j;

```

```

/* Allocate memory for LIS[] and initialize LIS values as 1 for
   all indexes */
int[] lis = new int[n];
for (i = 0; i < n; i++)
    lis[i] = 1;

/* Compute LIS values from left to right */
for (i = 1; i < n; i++)
    for (j = 0; j < i; j++)
        if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
            lis[i] = lis[j] + 1;

/* Allocate memory for lds and initialize LDS values for
   all indexes */
int[] lds = new int[n];
for (i = 0; i < n; i++)
    lds[i] = 1;

/* Compute LDS values from right to left */
for (i = n-2; i >= 0; i--)
    for (j = n-1; j > i; j--)
        if (arr[i] > arr[j] && lds[i] < lds[j] + 1)
            lds[i] = lds[j] + 1;

/* Return the maximum value of lis[i] + lds[i] - 1*/
int max = lis[0] + lds[0] - 1;
for (i = 1; i < n; i++)
    if (lis[i] + lds[i] - 1 > max)
        max = lis[i] + lds[i] - 1;

return max;
}

public static void main (String[] args)
{
    int arr[] = {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5,
                13, 3, 11, 7, 15};
    int n = arr.length;
    System.out.println("Length of LBS is "+ lbs( arr, n ));
}
}

```

[Run on IDE](#)

Python

```

# Dynamic Programming implementation of longest bitonic subsequence problem
"""

lbs() returns the length of the Longest Bitonic Subsequence in
arr[] of size n. The function mainly creates two temporary arrays
lis[] and lds[] and returns the maximum lis[i] + lds[i] - 1.

lis[i] ==> Longest Increasing subsequence ending with arr[i]
lds[i] ==> Longest decreasing subsequence starting with arr[i]
"""

def lbs(arr):
    n = len(arr)

    # allocate memory for LIS[] and initialize LIS values as 1
    # for all indexes
    lis = [1 for i in range(n+1)]

    # Compute LIS values from left to right
    for i in range(1, n):
        for j in range(0, i):
            if ((arr[i] > arr[j]) and (lis[i] < lis[j] + 1)):
                lis[i] = lis[j] + 1

```

```

# allocate memory for LDS and initialize LDS values for
# all indexes
lds = [1 for i in range(n+1)]

# Compute LDS values from right to left
for i in reversed(range(n-1)): #loop from n-2 downto 0
    for j in reversed(range(i-1 ,n)): #loop from n-1 downto i-1
        if(arr[i] > arr[j] and lds[i] < lds[j] + 1):
            lds[i] = lds[j] + 1

# Return the maximum value of (lis[i] + lds[i] - 1)
maximum = lis[0] + lds[0] + 1
for i in range(1 , n):
    maximum = max((lis[i] + lds[i]-1) , maximum)

return maximum

# Driver program to test the above function
arr = [0 , 8 , 4, 12, 2, 10 , 6 , 14 , 1 , 9 , 5 , 13,
       3, 11 , 7 , 15]
print "Length of LBS is",lbs(arr)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)

```

[Run on IDE](#)

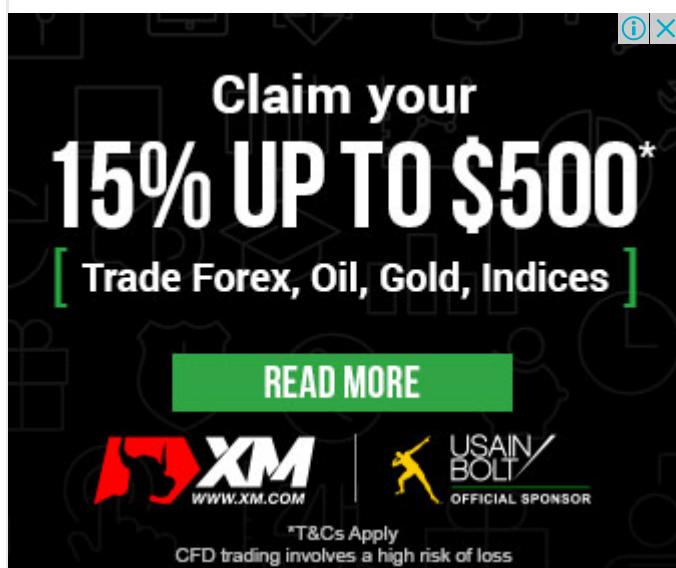
Output:

Length of LBS is 7

Time Complexity: O(n^2)

Auxiliary Space: O(n)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [subsequence](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.2

Average Difficulty : **3.2/5.0**
Based on **72** vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 16 (Floyd Warshall Algorithm)

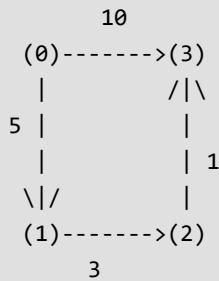
The [Floyd Warshall Algorithm](#) is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

Example:

Input:

```
graph[][] = { {0, 5, INF, 10},
              {INF, 0, 3, INF},
              {INF, INF, 0, 1},
              {INF, INF, INF, 0} }
```

which represents the following graph



Note that the value of $graph[i][j]$ is 0 if i is equal to j

And $graph[i][j]$ is INF (infinite) if there is no edge from vertex i to j .

Output:

Shortest distance matrix

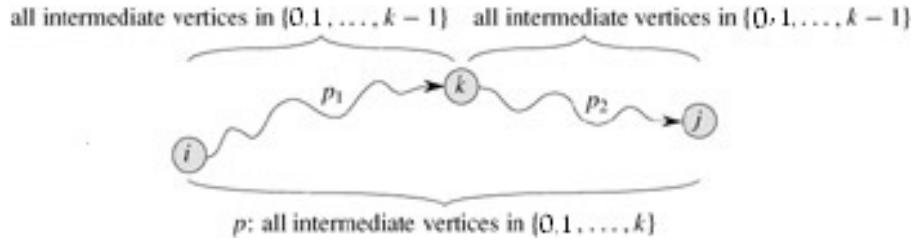
0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

Floyd Warshall Algorithm

We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices $\{0, 1, 2, \dots, k-1\}$ as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

- 1) k is not an intermediate vertex in shortest path from i to j . We keep the value of $dist[i][j]$ as it is.
- 2) k is an intermediate vertex in shortest path from i to j . We update the value of $dist[i][j]$ as $dist[i][k] + dist[k][j]$.

The following figure is taken from the Cormen book. It shows the above optimal substructure property in the all-pairs shortest path problem.



Following is implementations of the Floyd Warshall algorithm.

```
// C Program for Floyd Warshall Algorithm
#include<stdio.h>

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough value. This value will be used
   for vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][]);

// Solves the all-pairs shortest path problem using Floyd Warshall algorithm
void floydWarshall (int graph[][V])
{
    /* dist[][] will be the output matrix that will finally have the shortest
       distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix same as input graph matrix. Or
       we can say the initial values of shortest distances are based
       on shortest paths considering no intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to the set of intermediate vertices.
       ---> Before start of a iteration, we have shortest distances between all
             pairs of vertices such that the shortest distances consider only the
             vertices in set {0, 1, 2, .. k-1} as intermediate vertices.
       ----> After the end of a iteration, vertex no. k is added to the set of
             intermediate vertices and the set becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++)
            {
                // If vertex k is on the shortest path from
                // i to j, then update the value of dist[i][j]
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
        // Print the shortest distance matrix
        printSolution(dist);
    }

    /* A utility function to print solution */
}
```

```

void printSolution(int dist[][][V])
{
    printf ("Following matrix shows the shortest distances"
            " between every pair of vertices \n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf ("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

// driver program to test above function
int main()
{
    /* Let us create the following weighted graph
       10
       (0)----->(3)
       |           |
       5           1
       |           |
       \|          /\
       (1)----->(2)
       3           */
    int graph[V][V] = { {0, 5, INF, 10},
                        {INF, 0, 3, INF},
                        {INF, INF, 0, 1},
                        {INF, INF, INF, 0}
                      };
    // Print the solution
    floydWarshall(graph);
    return 0;
}

```

[Run on IDE](#)

Java

```

// A Java program for Floyd Warshall All Pairs Shortest
// Path algorithm.
import java.util.*;
import java.lang.*;
import java.io.*;

class AllPairShortestPath
{
    final static int INF = 99999, V = 4;

    void floydWarshall(int graph[][][])
    {
        int dist[][] = new int[V][V];
        int i, j, k;

        /* Initialize the solution matrix same as input graph matrix.
           Or we can say the initial values of shortest distances
           are based on shortest paths considering no intermediate
           vertex. */
        for (i = 0; i < V; i++)
            for (j = 0; j < V; j++)
                dist[i][j] = graph[i][j];

        /* Add all vertices one by one to the set of intermediate
           vertices.
           ---> Before start of a iteration, we have shortest

```

```

distances between all pairs of vertices such that
the shortest distances consider only the vertices in
set {0, 1, 2, .. k-1} as intermediate vertices.
----> After the end of a iteration, vertex no. k is added
      to the set of intermediate vertices and the set
      becomes {0, 1, 2, .. k} */
for (k = 0; k < V; k++)
{
    // Pick all vertices as source one by one
    for (i = 0; i < V; i++)
    {
        // Pick all vertices as destination for the
        // above picked source
        for (j = 0; j < V; j++)
        {
            // If vertex k is on the shortest path from
            // i to j, then update the value of dist[i][j]
            if (dist[i][k] + dist[k][j] < dist[i][j])
                dist[i][j] = dist[i][k] + dist[k][j];
        }
    }
}

// Print the shortest distance matrix
printSolution(dist);
}

void printSolution(int dist[][][])
{
    System.out.println("Following matrix shows the shortest "+
                        "distances between every pair of vertices");
    for (int i=0; i<V; ++i)
    {
        for (int j=0; j<V; ++j)
        {
            if (dist[i][j]==INF)
                System.out.print("INF ");
            else
                System.out.print(dist[i][j]+ " ");
        }
        System.out.println();
    }
}

// Driver program to test above function
public static void main (String[] args)
{
    /* Let us create the following weighted graph
       10
       (0)----->(3)
              /|\
              | |
              \ /|
       (1)----->(2)
              3   */
    int graph[][] = { {0, 5, INF, 10},
                     {INF, 0, 3, INF},
                     {INF, INF, 0, 1},
                     {INF, INF, INF, 0}
                 };
    AllPairShortestPath a = new AllPairShortestPath();

    // Print the solution
    a.floydWarshall(graph);
}

// Contributed by Aakash Hasija

```

Run on IDE

Python

```
# Python Program for Floyd Warshall Algorithm

# Number of vertices in the graph
V = 4

# Define infinity as the large enough value. This value will be
# used for vertices not connected to each other
INF = 99999

# Solves all pair shortest path via Floyd Warshall Algrorithm
def floydWarshall(graph):

    """ dist[][] will be the output matrix that will finally
        have the shortest distances between every pair of vertices """
    """ initializing the solution matrix same as input graph matrix
    OR we can say that the initial values of shortest distances
    are based on shortest paths considerting no
    intermediate vertices """
    dist = map(lambda i : map(lambda j : j , i) , graph)

    """ Add all vertices one by one to the set of intermediate
    vertices.
    ---> Before start of a iteration, we have shortest distances
    between all pairs of vertices such that the shortest
    distances consider only the vertices in set
    {0, 1, 2, .. k-1} as intermediate vertices.
    ----> After the end of a iteration, vertex no. k is
    added to the set of intermediate vertices and the
    set becomes {0, 1, 2, .. k}
    """
    for k in range(V):

        # pick all vertices as source one by one
        for i in range(V):

            # Pick all vertices as destination for the
            # above picked source
            for j in range(V):

                # If vertex k is on the shortest path from
                # i to j, then update the value of dist[i][j]
                dist[i][j] = min(dist[i][j] ,
                                  dist[i][k]+ dist[k][j]
                                 )
    printSolution(dist)

# A utility function to print the solution
def printSolution(dist):
    print "Following matrix shows the shortest distances\
between every pair of vertices"
    for i in range(V):
        for j in range(V):
            if(dist[i][j] == INF):
                print "%7s" %( "INF"),
            else:
                print "%7d\t" %(dist[i][j]),
            if j == V-1:
                print ""

# Driver program to test the above program
# Let us create the following weighted graph
"""

          10
          (0)----->(3)
          |           / \
          5           |   \
                      |     1
"""

```

```

\|/
(1)----->(2)
      3      """
graph = [[0,5,INF,10],
          [INF,0,3,INF],
          [INF, INF, 0, 1],
          [INF, INF, INF, 0]
        ]
# Print the solution
floydWarshall(graph);
# This code is contributed by Nikhil Kumar Singh(nickzuck_007)

```

[Run on IDE](#)

Output:

```

Following matrix shows the shortest distances between every pair of vertices
  0      5      8      9
INF      0      3      4
INF      INF     0      1
INF      INF     INF    0

```

Time Complexity: $O(V^3)$

The above program only prints the shortest distances. We can modify the solution to print the shortest paths also by storing the predecessor information in a separate 2D matrix.

Also, the value of INF can be taken as INT_MAX from limits.h to make sure that we handle maximum possible value. When we take INF as INT_MAX, we need to change the if condition in the above program to avoid arithmetic overflow.

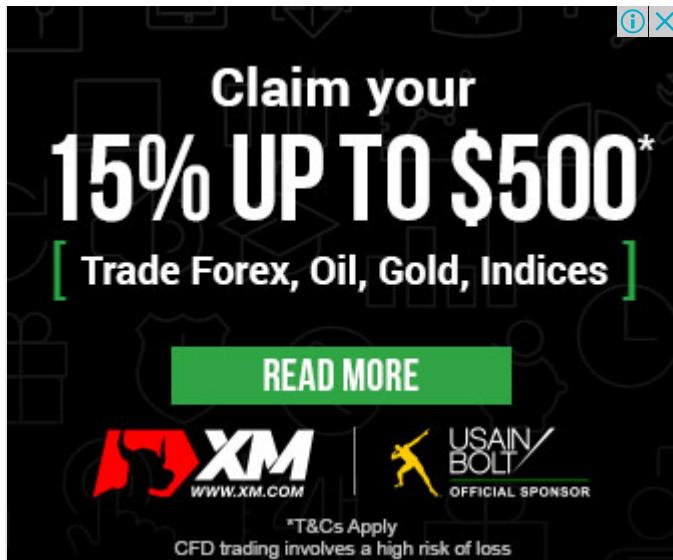
```

#include <limits.h>

#define INF INT_MAX
.....
if ( dist[i][k] != INF &&
    dist[k][j] != INF &&
    dist[i][k] + dist[k][j] < dist[i][j]
)
    dist[i][j] = dist[i][k] + dist[k][j];
.....

```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Graph](#) [shortest path](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.8

Average Difficulty : 2.8/5.0
Based on 65 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 17 (Palindrome Partitioning)

Given a string, a partitioning of the string is a *palindrome partitioning* if every substring of the partition is a palindrome. For example, “aba|b|bbabb|a|b|aba” is a palindrome partitioning of “ababbabbababa”. Determine the fewest cuts needed for palindrome partitioning of a given string. For example, minimum 3 cuts are needed for “ababbabbababa”. The three cuts are “a|babbbab|b|ababa”. If a string is palindrome, then minimum 0 cuts are needed. If a string of length n containing all different characters, then minimum n-1 cuts are needed.

String :	ababbabbababa
Palindrome partitioning:	a babbbab b ababa

Solution

This problem is a variation of [Matrix Chain Multiplication](#) problem. If the string is palindrome, then we simply return 0. Else, like the Matrix Chain Multiplication problem, we try making cuts at all possible places, recursively calculate the cost for each cut and return the minimum value.

Let the given string be str and minPalPartition() be the function that returns the fewest cuts needed for palindrome partitioning. following is the optimal substructure property.

```
// i is the starting index and j is the ending index. i must be passed as 0 and j as n-1
minPalPartition(str, i, j) = 0 if i == j. // When string is of length 1.
minPalPartition(str, i, j) = 0 if str[i..j] is palindrome.

// If none of the above conditions is true, then minPalPartition(str, i, j) can be
// calculated recursively using the following formula.
minPalPartition(str, i, j) = Min { minPalPartition(str, i, k) + 1 +
                                    minPalPartition(str, k+1, j) }
                                    where k varies from i to j-1
```

Following is Dynamic Programming solution. It stores the solutions to subproblems in two arrays P[][] and C[][] , and reuses the calculated values.

```
// Dynamic Programming Solution for Palindrome Partitioning Problem
#include <stdio.h>
#include <string.h>
#include <limits.h>
```

```

// A utility function to get minimum of two integers
int min (int a, int b) { return (a < b)? a : b; }

// Returns the minimum number of cuts needed to partition a string
// such that every part is a palindrome
int minPalPartion(char *str)
{
    // Get the length of the string
    int n = strlen(str);

    /* Create two arrays to build the solution in bottom up manner
       C[i][j] = Minimum number of cuts needed for palindrome partitioning
                 of substring str[i..j]
       P[i][j] = true if substring str[i..j] is palindrome, else false
                 Note that C[i][j] is 0 if P[i][j] is true */
    int C[n][n];
    bool P[n][n];

    int i, j, k, L; // different looping variables

    // Every substring of length 1 is a palindrome
    for (i=0; i<n; i++)
    {
        P[i][i] = true;
        C[i][i] = 0;
    }

    /* L is substring length. Build the solution in bottom up manner by
       considering all substrings of length starting from 2 to n.
       The loop structure is same as Matrix Chain Multiplication problem (
       See http://www.geeksforgeeks.org/archives/15553 )*/
    for (L=2; L<=n; L++)
    {
        // For substring of length L, set different possible starting indexes
        for (i=0; i<n-L+1; i++)
        {
            j = i+L-1; // Set ending index

            // If L is 2, then we just need to compare two characters. Else
            // need to check two corner characters and value of P[i+1][j-1]
            if (L == 2)
                P[i][j] = (str[i] == str[j]);
            else
                P[i][j] = (str[i] == str[j]) && P[i+1][j-1];

            // IF str[i..j] is palindrome, then C[i][j] is 0
            if (P[i][j] == true)
                C[i][j] = 0;
            else
            {
                // Make a cut at every possible location starting from i to j,
                // and get the minimum cost cut.
                C[i][j] = INT_MAX;
                for (k=i; k<=j-1; k++)
                    C[i][j] = min (C[i][j], C[i][k] + C[k+1][j]+1);
            }
        }
    }

    // Return the min cut value for complete string. i.e., str[0..n-1]
    return C[0][n-1];
}

// Driver program to test above function
int main()
{
    char str[] = "ababbabbababa";
    printf("Min cuts needed for Palindrome Partitioning is %d",
           minPalPartion(str));
    return 0;
}

```

Run on IDE

Output:

```
Min cuts needed for Palindrome Partitioning is 3
```

Time Complexity: $O(n^3)$

An optimization to above approach

In above approach, we can calculating minimum cut while finding all palindromic substring. If we finding all palindromic substring 1st and then we calculate minimum cut, time complexity will reduce to $O(n^2)$.

Thanks for **Vivek** for suggesting this optimization.

```
// Dynamic Programming Solution for Palindrome Partitioning Problem
#include <stdio.h>
#include <string.h>
#include <limits.h>

// A utility function to get minimum of two integers
int min (int a, int b) { return (a < b)? a : b; }

// Returns the minimum number of cuts needed to partition a string
// such that every part is a palindrome
int minPalPartion(char *str)
{
    // Get the length of the string
    int n = strlen(str);

    /* Create two arrays to build the solution in bottom up manner
       C[i] = Minimum number of cuts needed for palindrome partitioning
              of substring str[0..i]
       P[i][j] = true if substring str[i..j] is palindrome, else false
       Note that C[i] is 0 if P[0][i] is true */
    int C[n];
    bool P[n][n];

    int i, j, k, L; // different looping variables

    // Every substring of length 1 is a palindrome
    for (i=0; i<n; i++)
    {
        P[i][i] = true;
    }

    /* L is substring length. Build the solution in bottom up manner by
       considering all substrings of length starting from 2 to n. */
    for (L=2; L<=n; L++)
    {
        // For substring of length L, set different possible starting indexes
        for (i=0; i<n-L+1; i++)
        {
            j = i+L-1; // Set ending index

            // If L is 2, then we just need to compare two characters. Else
            // need to check two corner characters and value of P[i+1][j-1]
            if (L == 2)
                P[i][j] = (str[i] == str[j]);
            else
                P[i][j] = (str[i] == str[j]) && P[i+1][j-1];
        }
    }

    for (i=0; i<n; i++)
    {
        if (P[0][i] == true)
            C[i] = 0;
        else
        {
            C[i] = INT_MAX;
            for(j=0;j<i;j++)
            {
                if(P[j+1][i] == true && 1+C[j]<C[i])

```

```

        C[i]=1+C[j];
    }

}

// Return the min cut value for complete string. i.e., str[0..n-1]
return C[n-1];
}

// Driver program to test above function
int main()
{
    char str[] = "ababbabbababa";
    printf("Min cuts needed for Palindrome Partitioning is %d",
           minPalPartition(str));
    return 0;
}

```

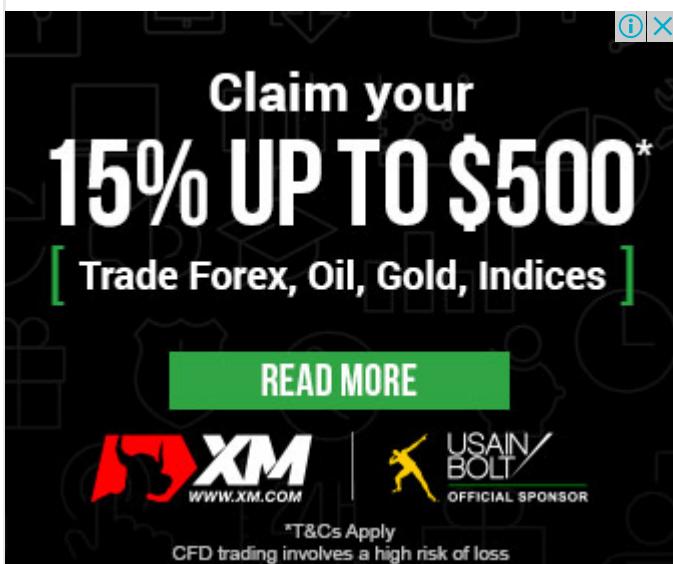
[Run on IDE](#)

Output:

Min cuts needed for Palindrome Partitioning is 3

Time Complexity: $O(n^2)$

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [palindrome](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix

- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.3

Average Difficulty : 4.3/5.0
Based on 87 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 18 (Partition problem)

Partition problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is same.

Examples

```
arr[] = {1, 5, 11, 5}
```

Output: true

The array can be partitioned as {1, 5, 5} and {11}

```
arr[] = {1, 5, 3}
```

Output: false

The array cannot be partitioned into equal sum sets.

We strongly recommend that you click here and practice it, before moving on to the solution.

Following are the two main steps to solve this problem:

- 1) Calculate sum of the array. If sum is odd, there can not be two subsets with equal sum, so return false.
- 2) If sum of array elements is even, calculate sum/2 and find a subset of array with sum equal to sum/2.

The first step is simple. The second step is crucial, it can be solved either using recursion or Dynamic Programming.

Recursive Solution

Following is the recursive property of the second step mentioned above.

```
Let isSubsetSum(arr, n, sum/2) be the function that returns true if
there is a subset of arr[0..n-1] with sum equal to sum/2
```

The isSubsetSum problem can be divided into two subproblems

- a) isSubsetSum() without considering last element

(reducing n to n-1)

- b) isSubsetSum considering the last element

(reducing sum/2 by arr[n-1] and n to n-1)

If any of the above the above subproblems return true, then return true.

```
isSubsetSum (arr, n, sum/2) = isSubsetSum (arr, n-1, sum/2) ||
                             isSubsetSum (arr, n-1, sum/2 - arr[n-1])
```

```
// A recursive C program for partition problem
#include <stdio.h>

// A utility function that returns true if there is
// a subset of arr[] with sum equal to given sum
bool isSubsetSum (int arr[], int n, int sum)
{
    // Base Cases
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;

    // If last element is greater than sum, then
    // ignore it
    if (arr[n-1] > sum)
        return isSubsetSum (arr, n-1, sum);

    /* else, check if sum can be obtained by any of
       the following
       (a) including the last element
       (b) excluding the last element
    */
    return isSubsetSum (arr, n-1, sum) ||
           isSubsetSum (arr, n-1, sum-arr[n-1]);
}

// Returns true if arr[] can be partitioned in two
// subsets of equal sum, otherwise false
bool findPartigion (int arr[], int n)
{
    // Calculate sum of the elements in array
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += arr[i];

    // If sum is odd, there cannot be two subsets
    // with equal sum
    if (sum%2 != 0)
        return false;

    // Find if there is subset with sum equal to
    // half of total sum
    return isSubsetSum (arr, n, sum/2);
}

// Driver program to test above function
int main()
{
    int arr[] = {3, 1, 5, 9, 12};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (findPartigion(arr, n) == true)
        printf("Can be divided into two subsets "
               "of equal sum");
    else
        printf("Can not be divided into two subsets"
               " of equal sum");
    return 0;
}
```

Run on IDE

Java

```

// A recursive Java solution for partition problem
import java.io.*;

class Partition
{
    // A utility function that returns true if there is a
    // subset of arr[] with sum equal to given sum
    static boolean isSubsetSum (int arr[], int n, int sum)
    {
        // Base Cases
        if (sum == 0)
            return true;
        if (n == 0 && sum != 0)
            return false;

        // If last element is greater than sum, then ignore it
        if (arr[n-1] > sum)
            return isSubsetSum (arr, n-1, sum);

        /* else, check if sum can be obtained by any of
           the following
           (a) including the last element
           (b) excluding the last element
        */
        return isSubsetSum (arr, n-1, sum) ||
               isSubsetSum (arr, n-1, sum-arr[n-1]);
    }

    // Returns true if arr[] can be partitioned in two
    // subsets of equal sum, otherwise false
    static boolean findPartition (int arr[], int n)
    {
        // Calculate sum of the elements in array
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += arr[i];

        // If sum is odd, there cannot be two subsets
        // with equal sum
        if (sum%2 != 0)
            return false;

        // Find if there is subset with sum equal to half
        // of total sum
        return isSubsetSum (arr, n, sum/2);
    }

    /*Driver function to check for above function*/
    public static void main (String[] args)
    {

        int arr[] = {3, 1, 5, 9, 12};
        int n = arr.length;
        if (findPartition(arr, n) == true)
            System.out.println("Can be divided into two "+
                               "subsets of equal sum");
        else
            System.out.println("Can not be divided into " +
                               "two subsets of equal sum");
    }
}
/* This code is contributed by Devesh Agrawal */

```

[Run on IDE](#)

Output:

Can be divided into two subsets of equal sum

Time Complexity: $O(2^n)$ In worst case, this solution tries two possibilities (whether to include or exclude) for every element.

Dynamic Programming Solution

The problem can be solved using dynamic programming when the sum of the elements is not too big. We can create a 2D array $\text{part}[][]$ of size $(\text{sum}/2)^*(n+1)$. And we can construct the solution in bottom up manner such that every filled entry has following property

```
part[i][j] = true if a subset of {arr[0], arr[1], ..arr[j-1]} has sum
            equal to i, otherwise false
```

```
// A Dynamic Programming based C program to partition problem
#include <stdio.h>
```

```
// Returns true if arr[] can be partitioned in two subsets of
// equal sum, otherwise false
bool findPartiion (int arr[], int n)
{
    int sum = 0;
    int i, j;

    // Caculate sum of all elements
    for (i = 0; i < n; i++)
        sum += arr[i];

    if (sum%2 != 0)
        return false;

    bool part[sum/2+1][n+1];

    // initialize top row as true
    for (i = 0; i <= n; i++)
        part[0][i] = true;

    // initialize leftmost column, except part[0][0], as 0
    for (i = 1; i <= sum/2; i++)
        part[i][0] = false;

    // Fill the partition table in bottom up manner
    for (i = 1; i <= sum/2; i++)
    {
        for (j = 1; j <= n; j++)
        {
            part[i][j] = part[i][j-1];
            if (i >= arr[j-1])
                part[i][j] = part[i][j] || part[i - arr[j-1]][j-1];
        }
    }

    /* // uncomment this part to print table
    for (i = 0; i <= sum/2; i++)
    {
        for (j = 0; j <= n; j++)
            printf ("%4d", part[i][j]);
        printf ("\n");
    } */

    return part[sum/2][n];
}

// Driver program to test above funtion
int main()
```

```

{
    int arr[] = {3, 1, 1, 2, 2, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (findPartition(arr, n) == true)
        printf("Can be divided into two subsets of equal sum");
    else
        printf("Can not be divided into two subsets of equal sum");
    getchar();
    return 0;
}

```

[Run on IDE](#)

Java

```

// A dynamic programming based Java program for partition problem
import java.io.*;

class Partition {

    // Returns true if arr[] can be partitioned in two subsets of
    // equal sum, otherwise false
    static boolean findPartition (int arr[], int n)
    {
        int sum = 0;
        int i, j;

        // Caculate sum of all elements
        for (i = 0; i < n; i++)
            sum += arr[i];

        if (sum%2 != 0)
            return false;

        boolean part[][]=new boolean[sum/2+1][n+1];

        // initialize top row as true
        for (i = 0; i <= n; i++)
            part[0][i] = true;

        // initialize leftmost column, except part[0][0], as 0
        for (i = 1; i <= sum/2; i++)
            part[i][0] = false;

        // Fill the partition table in bottom up manner
        for (i = 1; i <= sum/2; i++)
        {
            for (j = 1; j <= n; j++)
            {
                part[i][j] = part[i][j-1];
                if (i >= arr[j-1])
                    part[i][j] = part[i][j] ||
                                part[i - arr[j-1]][j-1];
            }
        }

        /* // uncomment this part to print table
        for (i = 0; i <= sum/2; i++)
        {
            for (j = 0; j <= n; j++)
                printf ("%4d", part[i][j]);
            printf ("\n");
        } */

        return part[sum/2][n];
    }

    /*Driver function to check for above function*/
    public static void main (String[] args)
    {
        int arr[] = {3, 1, 1, 2, 2, 1};

```

```

int n = arr.length;
if (findPartition(arr, n) == true)
    System.out.println("Can be divided into two "
                        "subsets of equal sum");
else
    System.out.println("Can not be divided into"
                        " two subsets of equal sum");
}
/* This code is contributed by Devesh Agrawal */

```

[Run on IDE](#)

Output:

Can be divided into two subsets of equal sum

Following diagram shows the values in partition table. The diagram is taken from the wiki page of partition problem.

The entry part[i][j] indicates whether there is a subset of {arr[0], arr[1], .. arr[j-1]} that sums to i

	{}	{3}	{3,1}	{3,1,1}	{3,1,1,2}	{3,1,1,2,2}	{3,1,1,2,2,1}
0	True	True	True	True	True	True	True
1	False	False	True	True	True	True	True
2	False	False	False	True	True	True	True
3	False	True	True	True	True	True	True
4	False	False	True	True	True	True	True
5	False	False	False	True	True	True	True

Dynamic Programming table for

arr[] = {3, 1, 1, 2, 2, 1}

Time Complexity: O(sum*n)

Auxiliary Space: O(sum*n)

Please note that this solution will not be feasible for arrays with big sum.

References:

http://en.wikipedia.org/wiki/Partition_problem

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 85 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 19 (Word Wrap Problem)

Given a sequence of words, and a limit on the number of characters that can be put in one line (line width). Put line breaks in the given sequence such that the lines are printed neatly. Assume that the length of each word is smaller than the line width.

The word processors like MS Word do task of placing line breaks. The idea is to have balanced lines. In other words, not have few lines with lots of extra spaces and some lines with small amount of extra spaces.

The extra spaces includes spaces put at the end of every line except the last one.

The problem is to minimize the following total cost.

Cost of a line = (Number of extra spaces in the line)³

Total Cost = Sum of costs for all lines

For example, consider the following string and line width M = 15

"Geeks for Geeks presents word wrap problem"

Following is the optimized arrangement of words in 3 lines

Geeks for Geeks

presents word

wrap problem

The total extra spaces in line 1, line 2 and line 3 are 0, 2 and 3 respectively.

So optimal value of total cost is $0 + 2^2 + 3^3 = 13$

Please note that the total cost function is not sum of extra spaces, but sum of cubes (or square is also used) of extra spaces. The idea behind this cost function is to balance the spaces among lines. For example, consider the following two arrangement of same set of words:

1) There are 3 lines. One line has 3 extra spaces and all other lines have 0 extra spaces. Total extra spaces = $3 + 0 + 0 = 3$. Total cost = $3^3 + 0^2 + 0^2 = 27$.

2) There are 3 lines. Each of the 3 lines has one extra space. Total extra spaces = $1 + 1 + 1 = 3$. Total cost = $1^3 + 1^3 + 1^3 = 3$.

Total extra spaces are 3 in both scenarios, but second arrangement should be preferred because extra spaces are balanced in all three lines. The cost function with cubic sum serves the purpose because the value of total cost in second scenario is less.

Method 1 (Greedy Solution)

The greedy solution is to place as many words as possible in the first line. Then do the same thing for the second

line and so on until all words are placed. This solution gives optimal solution for many cases, but doesn't give optimal solution in all cases. For example, consider the following string "aaa bb cc dddd" and line width as 6. Greedy method will produce following output.

```
aaa bb
cc
ddddd
```

Extra spaces in the above 3 lines are 0, 4 and 1 respectively. So total cost is $0 + 64 + 1 = 65$.

But the above solution is not the best solution. Following arrangement has more balanced spaces. Therefore less value of total cost function.

```
aaa
bb cc
ddddd
```

Extra spaces in the above 3 lines are 3, 1 and 1 respectively. So total cost is $27 + 1 + 1 = 29$.

Despite being sub-optimal in some cases, the greedy approach is used by many word processors like MS Word and OpenOffice.org Writer.

Method 2 (Dynamic Programming)

The following Dynamic approach strictly follows the algorithm given in solution of Cormen book. First we compute costs of all possible lines in a 2D table $lc[i][j]$. The value $lc[i][j]$ indicates the cost to put words from i to j in a single line where i and j are indexes of words in the input sequences. If a sequence of words from i to j cannot fit in a single line, then $lc[i][j]$ is considered infinite (to avoid it from being a part of the solution). Once we have the $lc[i][j]$ table constructed, we can calculate total cost using following recursive formula. In the following formula, $C[j]$ is the optimized total cost for arranging words from 1 to j .

$$c[j] = \begin{cases} 0 & \text{if } j = 0, \\ \min_{1 \leq i \leq j} (c[i-1] + lc[i, j]) & \text{if } j > 0. \end{cases}$$

The above recursion has [overlapping subproblem property](#). For example, the solution of subproblem $c(2)$ is used by $c(3)$, $C(4)$ and so on. So Dynamic Programming is used to store the results of subproblems. The array $c[]$ can be computed from left to right, since each value depends only on earlier values.

To print the output, we keep track of what words go on what lines, we can keep a parallel p array that points to where each c value came from. The last line starts at word $p[n]$ and goes through word n . The previous line starts at word $p[p[n]]$ and goes through word $p[n] - 1$, etc. The function `printSolution()` uses $p[]$ to print the solution.

In the below program, input is an array $l[]$ that represents lengths of words in a sequence. The value $l[i]$ indicates length of the i th word (i starts from 1) in the input sequence.

```
// A Dynamic programming solution for Word Wrap Problem
#include <limits.h>
#include <stdio.h>
#define INF INT_MAX

// A utility function to print the solution
int printSolution (int p[], int n);

// l[] represents lengths of different words in input sequence. For example,
// l[] = {3, 2, 2, 5} is for a sentence like "aaa bb cc dddd". n is size of
// l[] and M is line width (maximum no. of characters that can fit in a line)
void solveWordWrap (int l[], int n, int M)
{
```

```

// For simplicity, 1 extra space is used in all below arrays

// extras[i][j] will have number of extra spaces if words from i
// to j are put in a single line
int extras[n+1][n+1];

// lc[i][j] will have cost of a line which has words from
// i to j
int lc[n+1][n+1];

// c[i] will have total cost of optimal arrangement of words
// from 1 to i
int c[n+1];

// p[] is used to print the solution.
int p[n+1];

int i, j;

// calculate extra spaces in a single line. The value extra[i][j]
// indicates extra spaces if words from word number i to j are
// placed in a single line
for (i = 1; i <= n; i++)
{
    extras[i][i] = M - l[i-1];
    for (j = i+1; j <= n; j++)
        extras[i][j] = extras[i][j-1] - l[j-1] - 1;
}

// Calculate line cost corresponding to the above calculated extra
// spaces. The value lc[i][j] indicates cost of putting words from
// word number i to j in a single line
for (i = 1; i <= n; i++)
{
    for (j = i; j <= n; j++)
    {
        if (extras[i][j] < 0)
            lc[i][j] = INF;
        else if (j == n && extras[i][j] >= 0)
            lc[i][j] = 0;
        else
            lc[i][j] = extras[i][j]*extras[i][j];
    }
}

// Calculate minimum cost and find minimum cost arrangement.
// The value c[j] indicates optimized cost to arrange words
// from word number 1 to j.
c[0] = 0;
for (j = 1; j <= n; j++)
{
    c[j] = INF;
    for (i = 1; i <= j; i++)
    {
        if (c[i-1] != INF && lc[i][j] != INF && (c[i-1] + lc[i][j] < c[j]))
        {
            c[j] = c[i-1] + lc[i][j];
            p[j] = i;
        }
    }
}

printSolution(p, n);
}

int printSolution (int p[], int n)
{
    int k;
    if (p[n] == 1)
        k = 1;
    else
        k = printSolution (p, p[n]-1) + 1;
    printf ("Line number %d: From word no. %d to %d \n", k, p[n], n);
    return k;
}

```

```

}

// Driver program to test above functions
int main()
{
    int l[] = {3, 2, 2, 5};
    int n = sizeof(l)/sizeof(l[0]);
    int M = 6;
    solveWordWrap (l, n, M);
    return 0;
}

```

[Run on IDE](#)

Output:

```

Line number 1: From word no. 1 to 1
Line number 2: From word no. 2 to 3
Line number 3: From word no. 4 to 4

```

Time Complexity: $O(n^2)$

Auxiliary Space: $O(n^2)$ The auxiliary space used in the above program can be optimized to $O(n)$ (See the reference 2 for details)

References:

http://en.wikipedia.org/wiki/Word_wrap

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix

- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

4.4

Average Difficulty : **4.4/5.0**
Based on **73** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 20 (Maximum Length Chain of Pairs)

You are given n pairs of numbers. In every pair, the first number is always smaller than the second number. A pair (c, d) can follow another pair (a, b) if $b < c$. Chain of pairs can be formed in this fashion. Find the longest chain which can be formed from a given set of pairs. Source: [Amazon Interview | Set 2](#)

For example, if the given pairs are $\{(5, 24), (39, 60), (15, 28), (27, 40), (50, 90)\}$, then the longest chain that can be formed is of length 3, and the chain is $\{(5, 24), (27, 40), (50, 90)\}$

This problem is a variation of standard [Longest Increasing Subsequence](#) problem. Following is a simple two step process.

1) Sort given pairs in increasing order of first (or smaller) element.

2) Now run a modified LIS process where we compare the second element of already finalized LIS with the first element of new LIS being constructed.

The following code is a slight modification of method 2 of [this post](#).

```
#include<stdio.h>
#include<stdlib.h>

// Structure for a pair
struct pair
{
    int a;
    int b;
};

// This function assumes that arr[] is sorted in increasing order
// according the first (or smaller) values in pairs.
int maxChainLength( struct pair arr[], int n )
{
    int i, j, max = 0;
    int *mcl = (int*) malloc ( sizeof( int ) * n );

    /* Initialize MCL (max chain length) values for all indexes */
    for ( i = 0; i < n; i++ )
        mcl[i] = 1;

    /* Compute optimized chain length values in bottom up manner */
    for ( i = 1; i < n; i++ )
        for ( j = 0; j < i; j++ )
            if ( arr[i].a > arr[j].b && mcl[i] < mcl[j] + 1)
                mcl[i] = mcl[j] + 1;

    // mcl[i] now stores the maximum chain length ending with pair i

    /* Pick maximum of all MCL values */
    for ( i = 0; i < n; i++ )
        if ( max < mcl[i] )
```

```

max = mcl[i];

/* Free memory to avoid memory leak */
free( mcl );

return max;
}

/* Driver program to test above function */
int main()
{
    struct pair arr[] = { {5, 24}, {15, 25},
                          {27, 40}, {50, 60} };
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Length of maximum size chain is %d\n",
           maxChainLength( arr, n ));
    return 0;
}

```

[Run on IDE](#)

Output:

Length of maximum size chain is 3

Time Complexity: $O(n^2)$ where n is the number of pairs.

The given problem is also a variation of [Activity Selection problem](#) and can be solved in $(n \log n)$ time. To solve it as a activity selection problem, consider the first element of a pair as start time in activity selection problem, and the second element of pair as end time. Thanks to Palash for suggesting this approach.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#)

[Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 55 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

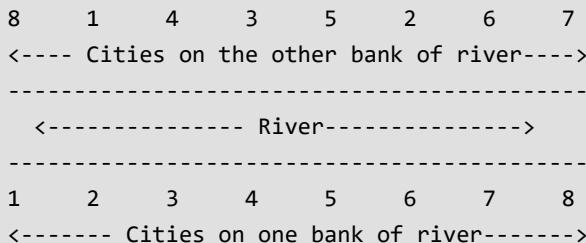


[Login/Register](#)

Dynamic Programming | Set 21 (Variations of LIS)

We have discussed Dynamic Programming solution for Longest Increasing Subsequence problem in [this](#) post and a $O(n \log n)$ solution in [this](#) post. Following are commonly asked variations of the standard [LIS problem](#).

1. Building Bridges: Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates $a(1) \dots a(n)$ and n cities on the northern bank with x-coordinates $b(1) \dots b(n)$. You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. When connecting cities, you can only connect city i on the northern bank to city i on the southern bank.



Source: [Dynamic Programming Practice Problems](#). The link also has well explained solution for the problem.

2. Maximum Sum Increasing Subsequence: Given an array of n positive integers. Write a program to find the maximum sum subsequence of the given array such that the integers in the subsequence are sorted in increasing order. For example, if input is {1, 101, 2, 3, 100, 4, 5}, then output should be {1, 2, 3, 100}. The solution to this problem has been published [here](#).

3. The Longest Chain You are given pairs of numbers. In a pair, the first number is smaller with respect to the second number. Suppose you have two sets (a, b) and (c, d) , the second set can follow the first set if $b < c$. So you can form a long chain in the similar fashion. Find the longest chain which can be formed. The solution to this problem has been published [here](#).

4. Box Stacking You are given a set of n types of rectangular 3-D boxes, where the i^{th} box has height $h(i)$, width $w(i)$ and depth $d(i)$ (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its

base. It is also allowable to use multiple instances of the same type of box.

Source: [Dynamic Programming Practice Problems](#). The link also has well explained solution for the problem.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [LIS](#) [subsequence](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **26** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

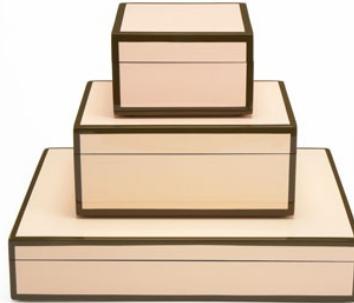


[Login/Register](#)

Dynamic Programming | Set 22 (Box Stacking Problem)

You are given a set of n types of rectangular 3-D boxes, where the i^{th} box has height $h(i)$, width $w(i)$ and depth $d(i)$ (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

Source: <http://people.csail.mit.edu/bdean/6.046/dp/>. The link also has video for explanation of solution.



The **Box Stacking problem** is a variation of **LIS** problem. We need to build a maximum height stack.

Following are the key points to note in the problem statement:

- 1) A box can be placed on top of another box only if both width and depth of the upper placed box are smaller than width and depth of the lower box respectively.
- 2) We can rotate boxes. For example, if there is a box with dimensions $\{1 \times 2 \times 3\}$ where 1 is height, 2×3 is base, then there can be three possibilities, $\{1 \times 2 \times 3\}$, $\{2 \times 1 \times 3\}$ and $\{3 \times 1 \times 2\}$.
- 3) We can use multiple instances of boxes. What it means is, we can have two different rotations of a box as part of our maximum height stack.

Following is the **solution** based on **DP** solution of **LIS** problem.

- 1) Generate all 3 rotations of all boxes. The size of rotation array becomes 3 times the size of original array. For simplicity, we consider depth as always smaller than or equal to width.
- 2) Sort the above generated $3n$ boxes in decreasing order of base area.
- 3) After sorting the boxes, the problem is same as LIS with following optimal substructure property.

$\text{MSH}(i) = \text{Maximum possible Stack Height with box } i \text{ at top of stack}$

$MSH(i) = \{ \text{Max} (MSH(j)) + \text{height}(i) \}$ where $j < i$ and $\text{width}(j) > \text{width}(i)$ and $\text{depth}(j) > \text{depth}(i)$.

If there is no such j then $MSH(i) = \text{height}(i)$

4) To get overall maximum height, we return $\max(MSH(i))$ where $0 < i < n$

Following is C++ implementation of the above solution.

```
/* Dynamic Programming implementation of Box Stacking problem */
#include<stdio.h>
#include<stdlib.h>

/* Representation of a box */
struct Box
{
    // h --> height, w --> width, d --> depth
    int h, w, d; // for simplicity of solution, always keep w <= d
};

// A utility function to get minimum of two integers
int min (int x, int y)
{ return (x < y)? x : y; }

// A utility function to get maximum of two integers
int max (int x, int y)
{ return (x > y)? x : y; }

/* Following function is needed for library function qsort(). We
   use qsort() to sort boxes in decreasing order of base area.
   Refer following link for help of qsort() and compare()
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare (const void *a, const void * b)
{
    return ( (*(Box *)b).d * (*(Box *)b).w ) -
            ( (*(Box *)a).d * (*(Box *)a).w );
}

/* Returns the height of the tallest stack that can be
   formed with give type of boxes */
int maxStackHeight( Box arr[], int n )
{
    /* Create an array of all rotations of given boxes
       For example, for a box {1, 2, 3}, we consider three
       instances{{1, 2, 3}, {2, 1, 3}, {3, 1, 2}} */
    Box rot[3*n];
    int index = 0;
    for (int i = 0; i < n; i++)
    {
        // Copy the original box
        rot[index] = arr[i];
        index++;

        // First rotation of box
        rot[index].h = arr[i].w;
        rot[index].d = max(arr[i].h, arr[i].d);
        rot[index].w = min(arr[i].h, arr[i].d);
        index++;

        // Second rotation of box
        rot[index].h = arr[i].d;
        rot[index].d = max(arr[i].h, arr[i].w);
        rot[index].w = min(arr[i].h, arr[i].w);
        index++;
    }

    // Now the number of boxes is 3n
    n = 3*n;

    /* Sort the array 'rot[]' in decreasing order, using library
       function for quick sort */
    qsort (rot, n, sizeof(rot[0]), compare);

    // Uncomment following two lines to print all rotations
    // for (int i = 0; i < n; i++)
}
```

```

//      printf("%d x %d x %d\n", rot[i].h, rot[i].w, rot[i].d);

/* Initialize msh values for all indexes
   msh[i] --> Maximum possible Stack Height with box i on top */
int msh[n];
for (int i = 0; i < n; i++)
    msh[i] = rot[i].h;

/* Compute optimized msh values in bottom up manner */
for (int i = 1; i < n; i++)
    for (int j = 0; j < i; j++)
        if ( rot[i].w < rot[j].w &&
            rot[i].d < rot[j].d &&
            msh[i] < msh[j] + rot[i].h
        )
        {
            msh[i] = msh[j] + rot[i].h;
        }

/* Pick maximum of all msh values */
int max = -1;
for ( int i = 0; i < n; i++ )
    if ( max < msh[i] )
        max = msh[i];

return max;
}

/* Driver program to test above function */
int main()
{
    Box arr[] = { {4, 6, 7}, {1, 2, 3}, {4, 5, 6}, {10, 12, 32} };
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("The maximum possible height of stack is %d\n",
           maxStackHeight (arr, n) );

    return 0;
}

```

[Run on IDE](#)

Output:

The maximum possible height of stack is 60

In the above program, given input boxes are {4, 6, 7}, {1, 2, 3}, {4, 5, 6}, {10, 12, 32}. Following are all rotations of the boxes in decreasing order of base area.

```

10 x 12 x 32
12 x 10 x 32
32 x 10 x 12
4 x 6 x 7
4 x 5 x 6
6 x 4 x 7
5 x 4 x 6
7 x 4 x 6
6 x 4 x 5
1 x 2 x 3
2 x 1 x 3
3 x 1 x 2

```

The height 60 is obtained by boxes { {3, 1, 2}, {1, 2, 3}, {6, 4, 5}, {4, 5, 6}, {4, 6, 7}, {32, 10, 12}, {10, 12, 32} }

Time Complexity: $O(n^2)$

Auxiliary Space: $O(n)$

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#)

[Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.7

Average Difficulty : 3.7/5.0
Based on 59 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Program for Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values

$$F_0 = 0 \text{ and } F_1 = 1.$$

We strongly recommend that you click here and practice it, before moving on to the solution.

Write a function `int fib(int n)` that returns F_n . For example, if $n = 0$, then `fib()` should return 0. If $n = 1$, then it should return 1. For $n > 1$, it should return $F_{n-1} + F_{n-2}$

Output:34

Following are different methods to get the nth Fibonacci number.

Method 1 (Use recursion)

A simple method that is a direct recursive implementation mathematical recurrence relation given above.

```
//Fibonacci Series using Recursion
#include<stdio.h>
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}

int main ()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
}
```

```
    return 0;
}
```

[Run on IDE](#)

Java

```
//Fibonacci Series using Recursion
class fibonacci
{
    static int fib(int n)
    {
        if (n <= 1)
            return n;
        return fib(n-1) + fib(n-2);
    }

    public static void main (String args[])
    {
        int n = 9;
        System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */
```

[Run on IDE](#)

Python

```
# Function for nth Fibonacci number

def Fibonacci(n):
    if n<0:
        print("Incorrect input")
    # First Fibonacci number is 0
    elif n==1:
        return 0
    # Second Fibonacci number is 1
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

# Driver Program

print(Fibonacci(9))

#This code is contributed by Saket Modi
```

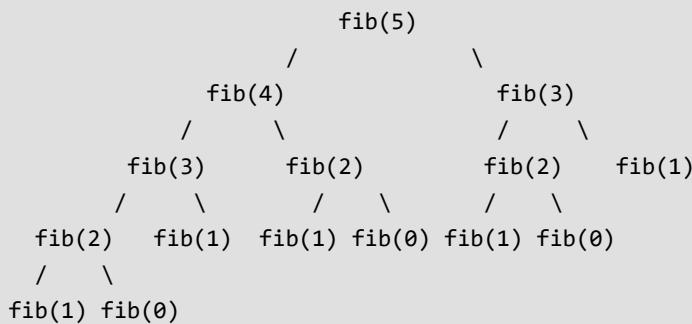
[Run on IDE](#)

Output

21

Time Complexity: $T(n) = T(n-1) + T(n-2)$ which is exponential.

We can observe that this implementation does a lot of repeated work (see the following recursion tree). So this is a bad implementation for nth Fibonacci number.



Extra Space: $O(n)$ if we consider the function call stack size, otherwise $O(1)$.

Method 2 (Use Dynamic Programming)

We can avoid the repeated work done in the method 1 by storing the Fibonacci numbers calculated so far.

```
//Fibonacci Series using Dynamic Programming
#include<stdio.h>

int fib(int n)
{
    /* Declare an array to store Fibonacci numbers. */
    int f[n+1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for (i = 2; i <= n; i++)
    {
        /* Add the previous 2 numbers in the series
           and store it */
        f[i] = f[i-1] + f[i-2];
    }

    return f[n];
}

int main ()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
    return 0;
}
```

Run on IDE

Java

```
// Fibonacci Series using Dynamic Programming
class fibonacci
{
    static int fib(int n)
    {
        /* Declare an array to store Fibonacci numbers. */
        int f[] = new int[n+1];
        int i;

        /* 0th and 1st number of the series are 0 and 1*/
        f[0] = 0;
        f[1] = 1;
```

```

for (i = 2; i <= n; i++)
{
    /* Add the previous 2 numbers in the series
       and store it */
    f[i] = f[i-1] + f[i-2];
}

return f[n];
}

public static void main (String args[])
{
    int n = 9;
    System.out.println(fib(n));
}
*/
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Python

```

# Function for nth fibonacci number - Dynamic Programming
# Taking 1st two fibonacci numbers as 0 and 1

FibArray = [0,1]

def fibonacci(n):
    if n<0:
        print("Incorrect input")
    elif n<=len(FibArray):
        return FibArray[n-1]
    else:
        temp_fib = fibonacci(n-1)+fibonacci(n-2)
        FibArray.append(temp_fib)
        return temp_fib

# Driver Program

print(fibonacci(9))

#This code is contributed by Saket Modi

```

Run on IDE

Output:

21

Time Complexity: O(n)

Extra Space: O(n)

Method 3 (Space Optimized Method 2)

We can optimize the space used in method 2 by storing the previous two numbers only because that is all we need to get the next Fibonacci number in series.

```

//Fibonacci Series using Space Optimized Method
#include<stdio.h>
int fib(int n)

```

```

{
    int a = 0, b = 1, c, i;
    if( n == 0)
        return a;
    for (i = 2; i <= n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

int main ()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
    return 0;
}

```

[Run on IDE](#)

Python

```

# Function for nth fibonacci number - Space Optimisataion
# Taking 1st two fibonacci numbers as 0 and 1

def fibonacci(n):
    a = 0
    b = 1
    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return a
    elif n == 1:
        return b
    else:
        for i in range(2,n):
            c = a + b
            a = b
            b = c
        return b

# Driver Program

print(fibonacci(9))

#This code is contributed by Saket Modi

```

[Run on IDE](#)

Time Complexity: O(n)

Extra Space: O(1)

Method 4 (Using power of the matrix {{1,1},{1,0}})

This another O(n) which relies on the fact that if we n times multiply the matrix $M = \begin{Bmatrix} 1 & 1 \\ 1 & 0 \end{Bmatrix}$ to itself (in other words calculate $\text{power}(M, n)$), then we get the $(n+1)$ th Fibonacci number as the element at row and column (0, 0) in the resultant matrix.

The matrix representation gives the following closed expression for the Fibonacci numbers:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

```
#include <stdio.h>

/* Helper function that multiplies 2 matrices F and M of size 2*2, and
   puts the multiplication result back to F[][] */
void multiply(int F[2][2], int M[2][2]);

/* Helper function that calculates F[][] raise to the power n and puts the
   result in F[][]
   Note that this function is designed only for fib() and won't work as general
   power function */
void power(int F[2][2], int n);

int fib(int n)
{
    int F[2][2] = {{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);

    return F[0][0];
}

void multiply(int F[2][2], int M[2][2])
{
    int x = F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y = F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z = F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w = F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

void power(int F[2][2], int n)
{
    int i;
    int M[2][2] = {{1,1},{1,0}};

    // n - 1 times multiply the matrix to {{1,0},{0,1}}
    for (i = 2; i <= n; i++)
        multiply(F, M);
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d", fib(n));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Java

```
class fibonacci
{
```

```

static int fib(int n)
{
    int F[][] = new int[][]{{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);

    return F[0][0];
}

/* Helper function that multiplies 2 matrices F and M of size 2*2, and
   puts the multiplication result back to F[][] */
static void multiply(int F[][], int M[][])
{
    int x = F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y = F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z = F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w = F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

/* Helper function that calculates F[][] raise to the power n and puts the
   result in F[][].
   Note that this function is designed only for fib() and won't work as general
   power function */
static void power(int F[][], int n)
{
    int i;
    int M[][] = new int[][]{{1,1},{1,0}};

    // n - 1 times multiply the matrix to {{1,0},{0,1}}
    for (i = 2; i <= n; i++)
        multiply(F, M);
}

/* Driver program to test above function */
public static void main (String args[])
{
    int n = 9;
    System.out.println(fib(n));
}
*/
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Time Complexity: O(n)

Extra Space: O(1)

Method 5 (Optimized Method 4)

The method 4 can be optimized to work in O(Logn) time complexity. We can do recursive multiplication to get power(M, n) in the previous method (Similar to the optimization done in [this](#) post)

```
#include <stdio.h>

void multiply(int F[2][2], int M[2][2]);

void power(int F[2][2], int n);
```

```

/* function that returns nth Fibonacci number */
int fib(int n)
{
    int F[2][2] = {{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);
    return F[0][0];
}

/* Optimized version of power() in method 4 */
void power(int F[2][2], int n)
{
    if( n == 0 || n == 1)
        return;
    int M[2][2] = {{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
}

void multiply(int F[2][2], int M[2][2])
{
    int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d", fib(9));
    getchar();
    return 0;
}

```

[Run on IDE](#)

Java

```

//Fibonacci Series using Optimized Method
class fibonacci
{
    /* function that returns nth Fibonacci number */
    static int fib(int n)
    {
        int F[][] = new int[][]{{1,1},{1,0}};
        if (n == 0)
            return 0;
        power(F, n-1);

        return F[0][0];
    }

    static void multiply(int F[][], int M[][])
    {
        int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
        int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
        int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
        int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];
    }
}

```

```

F[0][0] = x;
F[0][1] = y;
F[1][0] = z;
F[1][1] = w;
}

/* Optimized version of power() in method 4 */
static void power(int F[][], int n)
{
    if( n == 0 || n == 1)
        return;
    int M[][] = new int[][]{{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
}

/* Driver program to test above function */
public static void main (String args[])
{
    int n = 9;
    System.out.println(fib(n));
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Time Complexity: O(Logn)

Extra Space: O(Logn) if we consider the function call stack size, otherwise O(1).

Method 6 (O(Log n) Time)

Below is one more interesting recurrence formula that can be used to find n'th Fibonacci Number in O(Log n) time.

If n is even then k = n/2:
 $F(n) = [2*F(k-1) + F(k)]*F(k)$

If n is odd then k = (n + 1)/2
 $F(n) = F(k)*F(k) + F(k-1)*F(k-1)$

How does this formula work?

The formula can be derived from above matrix equation.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Taking determinant on both sides, we get

$$(-1)^n = F_{n+1}F_{n-1} - F_n^2$$

Moreover, since $A^n A^m = A^{n+m}$ for any square matrix A, the following identities can be derived (they are obtained from two different coefficients of the matrix product)

$$F_m F_n + F_{m-1} F_{n-1} = F_{m+n-1}$$

By putting n = n+1,

$$F_m F_{n+1} + F_{m-1} F_n = F_{m+n}$$

Putting $m = n$

$$F_{2n-1} = F_n^2 + F_{n-1}^2$$

$$F_{2n} = (F_{n-1} + F_{n+1})F_n = (2F_{n-1} + F_n)F_n \text{ (Source: } \text{Wiki})$$

To get the formula to be proved, we simply need to do following

If n is even, we can put $k = n/2$

If n is odd, we can put $k = (n+1)/2$

Below is C++ implementation of above idea.

```
// C++ Program to find n'th fibonacci Number in
// with O(Log n) arithmetic operations
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

// Create an array for memoization
int f[MAX] = {0};

// Returns n'th fuibonacci number using table f[]
int fib(int n)
{
    // Base cases
    if (n == 0)
        return 0;
    if (n == 1 || n == 2)
        return (f[n] = 1);

    // If fib(n) is already computed
    if (f[n])
        return f[n];

    int k = (n & 1)? (n+1)/2 : n/2;

    // Applying above formula [Note value n&1 is 1
    // if n is odd, else 0.
    f[n] = (n & 1)? (fib(k)*fib(k) + fib(k-1)*fib(k-1))
                  : (2*fib(k-1) + fib(k))*fib(k);

    return f[n];
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d ", fib(n));
    return 0;
}
```

Run on IDE

Output :

34

Time complexity of this solution is $O(\log n)$ as we divide the problem to half in every recursive call.

This method is contributed by Chirag Agarwal.

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

References:

http://en.wikipedia.org/wiki/Fibonacci_number
<http://www.ics.uci.edu/~eppstein/161/960109.html>



Live There with Airbnb

From bedrooms to apartments to villas -
Airbnb has it all.

airbnb.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Mathematical](#) [Dynamic Programming](#) [Fibonacci](#) [MathematicalAlgo](#) [series](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.1

Average Difficulty : 3.1/5.0
Based on 83 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Minimum number of jumps to reach end

Given an array of integers where each element represents the max number of steps that can be made forward from that element. Write a function to return the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then cannot move through that element.

Example:

```
Input: arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}
Output: 3 (1-> 3 -> 8 ->9)
```

First element is 1, so can only go to 3. Second element is 3, so can make at most 3 steps eg to 5 or 8 or 9.

We strongly recommend that you click here and practice it, before moving on to the solution.

Method 1 (Naive Recursive Approach)

A naive approach is to start from the first element and recursively call for all the elements reachable from first element. The minimum number of jumps to reach end from first can be calculated using minimum number of jumps needed to reach end from the elements reachable from first.

$\text{minJumps}(\text{start}, \text{end}) = \text{Min} (\text{minJumps}(k, \text{end})) \text{ for all } k \text{ reachable from start}$

```
#include <stdio.h>
#include <limits.h>

// Returns minimum number of jumps to reach arr[h] from arr[1]
int minJumps(int arr[], int l, int h)
{
    // Base case: when source and destination are same
    if (h == l)
        return 0;

    // When nothing is reachable from the given source
    if (arr[l] == 0)
        return INT_MAX;

    // Traverse through all the points reachable from arr[l]. Recursively
    // get the minimum number of jumps needed to reach arr[h] from these
    // reachable points.
    int min = INT_MAX;
    for (int i = l+1; i <= h && i <= l + arr[l]; i++)
    {
        int jumps = minJumps(arr, i, h);
        if(jumps != INT_MAX && jumps + 1 < min)
```

```

        min = jumps + 1;
    }

    return min;
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 3, 6, 3, 2, 3, 6, 8, 9, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Minimum number of jumps to reach end is %d ", minJumps(arr, 0, n-1));
    return 0;
}

```

[Run on IDE](#)

If we trace the execution of this method, we can see that there will be overlapping subproblems. For example, `minJumps(3, 9)` will be called two times as `arr[3]` is reachable from `arr[1]` and `arr[2]`. So this problem has both properties ([optimal substructure](#) and [overlapping subproblems](#)) of Dynamic Programming.

Method 2 (Dynamic Programming)

In this method, we build a `jumps[]` array from left to right such that `jumps[i]` indicates the minimum number of jumps needed to reach `arr[i]` from `arr[0]`. Finally, we return `jumps[n-1]`.

```

#include <stdio.h>
#include <limits.h>

int min(int x, int y) { return (x < y)? x: y; }

// Returns minimum number of jumps to reach arr[n-1] from arr[0]
int minJumps(int arr[], int n)
{
    int *jumps = new int[n]; // jumps[n-1] will hold the result
    int i, j;

    if (n == 0 || arr[0] == 0)
        return INT_MAX;

    jumps[0] = 0;

    // Find the minimum number of jumps to reach arr[i]
    // from arr[0], and assign this value to jumps[i]
    for (i = 1; i < n; i++)
    {
        jumps[i] = INT_MAX;
        for (j = 0; j < i; j++)
        {
            if (i <= j + arr[j] && jumps[j] != INT_MAX)
            {
                jumps[i] = min(jumps[i], jumps[j] + 1);
                break;
            }
        }
    }
    return jumps[n-1];
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 3, 6, 1, 0, 9};
    int size = sizeof(arr)/sizeof(int);
    printf("Minimum number of jumps to reach end is %d ", minJumps(arr,size));
    return 0;
}

```

[Run on IDE](#)

Output:

```
Minimum number of jumps to reach end is 3
```

Thanks to [paras](#) for suggesting this method.

Time Complexity: $O(n^2)$

Method 3 (Dynamic Programming)

In this method, we build jumps[] array from right to left such that jumps[i] indicates the minimum number of jumps needed to reach arr[n-1] from arr[i]. Finally, we return arr[0].

```
int minJumps(int arr[], int n)
{
    int *jumps = new int[n]; // jumps[0] will hold the result
    int min;

    // Minimum number of jumps needed to reach last element
    // from last elements itself is always 0
    jumps[n-1] = 0;

    int i, j;

    // Start from the second element, move from right to left
    // and construct the jumps[] array where jumps[i] represents
    // minimum number of jumps needed to reach arr[m-1] from arr[i]
    for (i = n-2; i >=0; i--)
    {
        // If arr[i] is 0 then arr[n-1] can't be reached from here
        if (arr[i] == 0)
            jumps[i] = INT_MAX;

        // If we can directly reach to the end point from here then
        // jumps[i] is 1
        else if (arr[i] >= n - i - 1)
            jumps[i] = 1;

        // Otherwise, to find out the minimum number of jumps needed
        // to reach arr[n-1], check all the points reachable from here
        // and jumps[] value for those points
        else
        {
            min = INT_MAX; // initialize min value

            // following loop checks with all reachable points and
            // takes the minimum
            for (j = i+1; j < n && j <= arr[i] + i; j++)
            {
                if (min > jumps[j])
                    min = jumps[j];
            }

            // Handle overflow
            if (min != INT_MAX)
                jumps[i] = min + 1;
            else
                jumps[i] = min; // or INT_MAX
        }
    }

    return jumps[0];
}
```

Run on IDE

Time Complexity: $O(n^2)$ in worst case.

Thanks to [Ashish](#) for suggesting this solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.8 Average Difficulty : 2.8/5.0
Based on 99 vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Maximum size square sub-matrix with all 1s

Given a binary matrix, find out the maximum size square sub-matrix with all 1s.

For example, consider the below binary matrix.

```

0 1 1 0 1
1 1 0 1 0
0 1 1 1 0
1 1 1 1 0
1 1 1 1 1
0 0 0 0 0
    
```

The maximum square sub-matrix with all set bits is

```

1 1 1
1 1 1
1 1 1
    
```

We strongly recommend that you click [here](#) and practice it, before moving on to the solution.

Algorithm:

Let the given binary matrix be $M[R][C]$. The idea of the algorithm is to construct an auxiliary size matrix $S[][]$ in which each entry $S[i][j]$ represents size of the square sub-matrix with all 1s including $M[i][j]$ where $M[i][j]$ is the rightmost and bottommost entry in sub-matrix.

- 1) Construct a sum matrix $S[R][C]$ for the given $M[R][C]$.
 - a) Copy first row and first columns as it is from $M[][]$ to $S[][]$
 - b) For other entries, use following expressions to construct $S[][]$

```

        If M[i][j] is 1 then
            S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1
        Else /*If M[i][j] is 0*/
            S[i][j] = 0
      
```
- 2) Find the maximum entry in $S[R][C]$
- 3) Using the value and coordinates of maximum entry in $S[i]$, print sub-matrix of $M[][]$

For the given M[R][C] in above example, constructed S[R][C] would be:

```

0 1 1 0 1
1 1 0 1 0
0 1 1 1 0
1 1 2 2 0
1 2 2 3 1
0 0 0 0 0

```

The value of maximum entry in above matrix is 3 and coordinates of the entry are (4, 3). Using the maximum value and its coordinates, we can find out the required sub-matrix.

```

#include<stdio.h>
#define bool int
#define R 6
#define C 5

void printMaxSubSquare(bool M[R][C])
{
    int i,j;
    int S[R][C];
    int max_of_s, max_i, max_j;

    /* Set first column of S[][]*/
    for(i = 0; i < R; i++)
        S[i][0] = M[i][0];

    /* Set first row of S[][]*/
    for(j = 0; j < C; j++)
        S[0][j] = M[0][j];

    /* Construct other entries of S[][]*/
    for(i = 1; i < R; i++)
    {
        for(j = 1; j < C; j++)
        {
            if(M[i][j] == 1)
                S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1;
            else
                S[i][j] = 0;
        }
    }

    /* Find the maximum entry, and indexes of maximum entry
     * in S[][] */
    max_of_s = S[0][0]; max_i = 0; max_j = 0;
    for(i = 0; i < R; i++)
    {
        for(j = 0; j < C; j++)
        {
            if(max_of_s < S[i][j])
            {
                max_of_s = S[i][j];
                max_i = i;
                max_j = j;
            }
        }
    }

    printf("\n Maximum size sub-matrix is: \n");
    for(i = max_i; i > max_i - max_of_s; i--)
    {
        for(j = max_j; j > max_j - max_of_s; j--)
        {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
}

```

```

/* UTILITY FUNCTIONS */
/* Function to get minimum of three values */
int min(int a, int b, int c)
{
    int m = a;
    if (m > b)
        m = b;
    if (m > c)
        m = c;
    return m;
}

/* Driver function to test above functions */
int main()
{
    bool M[R][C] = {{0, 1, 1, 0, 1},
                    {1, 1, 0, 1, 0},
                    {0, 1, 1, 1, 0},
                    {1, 1, 1, 1, 0},
                    {1, 1, 1, 1, 1},
                    {0, 0, 0, 0, 0}};

    printMaxSubSquare(M);
    getchar();
}

```

[Run on IDE](#)

Time Complexity: O(m*n) where m is number of rows and n is number of columns in the given matrix.

Auxiliary Space: O(m*n) where m is number of rows and n is number of columns in the given matrix.

Algorithmic Paradigm: Dynamic Programming

Please write comments if you find any bug in above code/algorithm, or find other ways to solve the same problem



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Matrix](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +

- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 114 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Ugly Numbers

Ugly numbers are numbers whose only prime factors are 2, 3 or 5. The sequence

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ...

shows the first 11 ugly numbers. By convention, 1 is included.

Write a program to find and print the 150'th ugly number.

We strongly recommend that you click here and practice it, before moving on to the solution.

METHOD 1 (Simple)

Thanks to [Nedylko Draganov](#) for suggesting this solution.

Algorithm:

Loop for all positive integers until ugly number count is smaller than n, if an integer is ugly than increment ugly number count.

To check if a number is ugly, divide the number by greatest divisible powers of 2, 3 and 5, if the number becomes 1 then it is an ugly number otherwise not.

For example, let us see how to check for 300 is ugly or not. Greatest divisible power of 2 is 4, after dividing 300 by 4 we get 75. Greatest divisible power of 3 is 3, after dividing 75 by 3 we get 25. Greatest divisible power of 5 is 25, after dividing 25 by 25 we get 1. Since we get 1 finally, 300 is ugly number.

Implementation:

```
# include<stdio.h>
# include<stdlib.h>

/*This function divides a by greatest divisible
 power of b*/
int maxDivide(int a, int b)
{
    while (a%b == 0)
        a = a/b;
    return a;
}

/* Function to check if a number is ugly or not */
int isUgly(int no)
```

```

{
    no = maxDivide(no, 2);
    no = maxDivide(no, 3);
    no = maxDivide(no, 5);

    return (no == 1)? 1 : 0;
}

/* Function to get the nth ugly number*/
int getNthUglyNo(int n)
{
    int i = 1;
    int count = 1; /* ugly number count */

    /*Check for all integers untill ugly count
       becomes n*/
    while (n > count)
    {
        i++;
        if (isUgly(i))
            count++;
    }
    return i;
}

/* Driver program to test above functions */
int main()
{
    unsigned no = getNthUglyNo(150);
    printf("150th ugly no. is %d ", no);
    getchar();
    return 0;
}

```

[Run on IDE](#)

This method is not time efficient as it checks for all integers until ugly number count becomes n, but space complexity of this method is O(1)

METHOD 2 (Use Dynamic Programming)

Here is a time efficient solution with O(n) extra space. The ugly-number sequence is 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ...

because every number can only be divided by 2, 3, 5, one way to look at the sequence is to split the sequence to three groups as below:

- (1) 1×2, 2×2, 3×2, 4×2, 5×2, ...
- (2) 1×3, 2×3, 3×3, 4×3, 5×3, ...
- (3) 1×5, 2×5, 3×5, 4×5, 5×5, ...

We can find that every subsequence is the ugly-sequence itself (1, 2, 3, 4, 5, ...) multiply 2, 3, 5. Then we use similar merge method as merge sort, to get every ugly number from the three subsequence. Every step we choose the smallest one, and move one step after.

Algorithm:

- 1 Declare an array for ugly numbers: ugly[150]
- 2 Initialize first ugly no: ugly[0] = 1
- 3 Initialize three array index variables i2, i3, i5 to point to 1st element of the ugly array:
i2 = i3 = i5 = 0;
- 4 Initialize 3 choices for the next ugly no:
next_multiple_of_2 = ugly[i2]*2;
next_multiple_of_3 = ugly[i3]*3

```

next_multiple_of_5 = ugly[i5]*5;
5 Now go in a loop to fill all ugly numbers till 150:
For (i = 1; i < 150; i++)
{
/* These small steps are not optimized for good
readability. Will optimize them in C program */
next_ugly_no = Min(next_multiple_of_2,
                    next_multiple_of_3,
                    next_multiple_of_5);
if (next_ugly_no == next_multiple_of_2)
{
    i2 = i2 + 1;
    next_multiple_of_2 = ugly[i2]*2;
}
if (next_ugly_no == next_multiple_of_3)
{
    i3 = i3 + 1;
    next_multiple_of_3 = ugly[i3]*3;
}
if (next_ugly_no == next_multiple_of_5)
{
    i5 = i5 + 1;
    next_multiple_of_5 = ugly[i5]*5;
}
ugly[i] = next_ugly_no
}/* end of for loop */
6.return next_ugly_no

```

Example:

Let us see how it works

```

initialize
ugly[] = | 1 |
i2 = i3 = i5 = 0;

```

First iteration

```

ugly[1] = Min(ugly[i2]*2, ugly[i3]*3, ugly[i5]*5)
        = Min(2, 3, 5)
        = 2
ugly[] = | 1 | 2 |
i2 = 1, i3 = i5 = 0 (i2 got incremented )

```

Second iteration

```

ugly[2] = Min(ugly[i2]*2, ugly[i3]*3, ugly[i5]*5)
        = Min(4, 3, 5)
        = 3
ugly[] = | 1 | 2 | 3 |
i2 = 1, i3 = 1, i5 = 0 (i3 got incremented )

```

Third iteration

```

ugly[3] = Min(ugly[i2]*2, ugly[i3]*3, ugly[i5]*5)
        = Min(4, 6, 5)
        = 4
ugly[] = | 1 | 2 | 3 | 4 |
i2 = 2, i3 = 1, i5 = 0 (i2 got incremented )

```

Fourth iteration

```

ugly[4] = Min(ugly[i2]*2, ugly[i3]*3, ugly[i5]*5)
        = Min(6, 6, 5)
        = 5
ugly[] = | 1 | 2 | 3 | 4 | 5 |
i2 = 2, i3 = 1, i5 = 1 (i5 got incremented )

```

Fifth iteration

```

ugly[4] = Min(ugly[i2]*2, ugly[i3]*3, ugly[i5]*5)
        = Min(6, 6, 10)
        = 6
ugly[] = | 1 | 2 | 3 | 4 | 5 | 6 |
i2 = 3, i3 = 2, i5 = 1 (i2 and i3 got incremented )

```

Will continue same way till $i < 150$

Program:

```

# include<stdio.h>
# include<stdlib.h>
# define bool int

/* Function to find minimum of 3 numbers */
unsigned min(unsigned , unsigned , unsigned );

/* Function to get the nth ugly number*/
unsigned getNthUglyNo(unsigned n)
{
    unsigned *ugly =
        (unsigned *)(malloc (sizeof(unsigned)*n));
    unsigned i2 = 0, i3 = 0, i5 = 0;
    unsigned i;
    unsigned next_multiple_of_2 = 2;
    unsigned next_multiple_of_3 = 3;
    unsigned next_multiple_of_5 = 5;
    unsigned next_ugly_no = 1;
    *(ugly+0) = 1;

    for(i=1; i<n; i++)
    {
        next_ugly_no = min(next_multiple_of_2,
                            next_multiple_of_3,
                            next_multiple_of_5);
        *(ugly+i) = next_ugly_no;
        if(next_ugly_no == next_multiple_of_2)
        {
            i2 = i2+1;
            next_multiple_of_2 = *(ugly+i2)*2;
        }
        if(next_ugly_no == next_multiple_of_3)
        {
            i3 = i3+1;
            next_multiple_of_3 = *(ugly+i3)*3;
        }
        if(next_ugly_no == next_multiple_of_5)
        {
            i5 = i5+1;
            next_multiple_of_5 = *(ugly+i5)*5;
        }
    } /*End of for loop (i=1; i<n; i++) */
    return next_ugly_no;
}

/* Function to find minimum of 3 numbers */
unsigned min(unsigned a, unsigned b, unsigned c)
{
    if(a <= b)
    {
        if(a <= c)
            return a;
    }
}

```

```

        else
            return c;
    }
    if(b <= c)
        return b;
    else
        return c;
}

/* Driver program to test above functions */
int main()
{
    unsigned no = getNthUglyNo(150);
    printf("%dth ugly no. is %d ", 150, no);
    getchar();
    return 0;
}

```

[Run on IDE](#)

Algorithmic Paradigm: Dynamic Programming

Time Complexity: O(n)

Storage Complexity: O(n)

Please write comments if you find any bug in the above program or other ways to solve the same problem.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence

- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 81 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Largest Sum Contiguous Subarray

Write an efficient C program to find the sum of contiguous subarray within a one-dimensional array of numbers which has the largest sum.

Largest Subarray Sum Problem

-2	-3	4	-1	-2	1	5	-3
0	1	2	3	4	5	6	7

$$4 + (-1) + (-2) + 1 + 5 = 7$$

Maximum Contiguous Array Sum is 7

We strongly recommend that you click here and practice it, before moving on to the solution.

Kadane's Algorithm:

```

Initialize:
max_so_far = 0
max_ending_here = 0

Loop for each element of the array
(a) max_ending_here = max_ending_here + a[i]
(b) if(max_ending_here < 0)
    max_ending_here = 0
(c) if(max_so_far < max_ending_here)
    max_so_far = max_ending_here
return max_so_far

```

Explanation:

Simple idea of the Kadane's algorithm is to look for all positive contiguous segments of the array (max_ending_here is used for this). And keep track of maximum sum contiguous segment among all positive segments (max_so_far is used for this). Each time we get a positive sum compare it with max_so_far and update max_so_far if it is greater than max_so_far

Lets take the example:

```
{-2, -3, 4, -1, -2, 1, 5, -3}
```

```
max_so_far = max_ending_here = 0
```

```
for i=0, a[0] = -2
```

```
max_ending_here = max_ending_here + (-2)
```

Set max_ending_here = 0 because max_ending_here < 0

```
for i=1, a[1] = -3
```

```
max_ending_here = max_ending_here + (-3)
```

Set max_ending_here = 0 because max_ending_here < 0

```
for i=2, a[2] = 4
```

```
max_ending_here = max_ending_here + (4)
```

```
max_ending_here = 4
```

max_so_far is updated to 4 because max_ending_here greater than max_so_far which was 0 till now

```
for i=3, a[3] = -1
```

```
max_ending_here = max_ending_here + (-1)
```

```
max_ending_here = 3
```

```
for i=4, a[4] = -2
```

```
max_ending_here = max_ending_here + (-2)
```

```
max_ending_here = 1
```

```
for i=5, a[5] = 1
```

```
max_ending_here = max_ending_here + (1)
```

```
max_ending_here = 2
```

```
for i=6, a[6] = 5
```

```
max_ending_here = max_ending_here + (5)
```

```
max_ending_here = 7
```

max_so_far is updated to 7 because max_ending_here is greater than max_so_far

```
for i=7, a[7] = -3
```

```
max_ending_here = max_ending_here + (-3)
```

```
max_ending_here = 4
```

Program:

```
// C++ program to print largest contiguous array sum
#include<iostream>
using namespace std;

int maxSubArraySum(int a[], int size)
```

```

{
    int max_so_far = 0, max_ending_here = 0;

    for (int i = 0; i < size; i++)
    {
        max_ending_here = max_ending_here + a[i];
        if (max_ending_here < 0)
            max_ending_here = 0;
        if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;
    }
    return max_so_far;
}

/*Driver program to test maxSubArraySum*/
int main()
{
    int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};
    int n = sizeof(a)/sizeof(a[0]);
    int max_sum = maxSubArraySum(a, n);
    cout << "Maximum contiguous sum is " << max_sum;
    return 0;
}

```

[Run on IDE](#)

Java

```

import java.io.*;
// Java program to print largest contiguous array sum
import java.util.*;

class Kadane
{
    public static void main (String[] args)
    {
        int [] a = {-2, -3, 4, -1, -2, 1, 5, -3};
        System.out.println("Maximum contiguous sum is " +
                           maxSubArraySum(a));
    }

    static int maxSubArraySum(int a[])
    {
        int size = a.length;
        int max_so_far = 0, max_ending_here = 0;

        for (int i = 0; i < size; i++)
        {
            max_ending_here = max_ending_here + a[i];
            if (max_ending_here < 0)
                max_ending_here = 0;
            if (max_so_far < max_ending_here)
                max_so_far = max_ending_here;
        }
        return max_so_far;
    }
}

```

[Run on IDE](#)

Python

```

# Python program to find maximum contiguous subarray

# Function to find the maximum contiguous subarray
def maxSubArraySum(a,size):

    max_so_far = 0

```

```

max_ending_here = 0

for i in range(0, size):
    max_ending_here = max_ending_here + a[i]
    if max_ending_here < 0:
        max_ending_here = 0

    if (max_so_far < max_ending_here):
        max_so_far = max_ending_here

return max_so_far

# Driver function to check the above function
a = [-2, -3, 4, -1, -2, 1, 5, -3]
print("Maximum contiguous sum is", maxSubArraySum(a, len(a)))

```

#This code is contributed by _Devesh Agrawal_

[Run on IDE](#)

Output:

Maximum contiguous sum is 7

Notes:

Algorithm doesn't work for all negative numbers. It simply returns 0 if all numbers are negative. For handling this we can add an extra phase before actual implementation. The phase will look if all numbers are negative, if they are it will return maximum of them (or smallest in terms of absolute value). There may be other ways to handle it though.

Above program can be optimized further, if we compare max_so_far with max_ending_here only if max_ending_here is greater than 0.

```

int maxSubArraySum(int a[], int size)
{
    int max_so_far = 0, max_ending_here = 0;
    for (int i = 0; i < size; i++)
    {
        max_ending_here = max_ending_here + a[i];
        if (max_ending_here < 0)
            max_ending_here = 0;

        /* Do not compare for all elements. Compare only
           when max_ending_here > 0 */
        else if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;
    }
    return max_so_far;
}

```

[Run on IDE](#)

Python

```

def maxSubArraySum(a, size):

    max_so_far = 0
    max_ending_here = 0

```

```

for i in range(0, size):
    max_ending_here = max_ending_here + a[i]
    if max_ending_here < 0:
        max_ending_here = 0

    # Do not compare for all elements. Compare only
    # when max_ending_here > 0
    elif (max_so_far < max_ending_here):
        max_so_far = max_ending_here

return max_so_far

```

[Run on IDE](#)

Time Complexity: O(n)

Algorithmic Paradigm: Dynamic Programming

Following is another simple implementation suggested by **Mohit Kumar**. The implementation handles the case when all numbers in array are negative.

```

#include<iostream>
using namespace std;

int maxSubArraySum(int a[], int size)
{
    int max_so_far = a[0];
    int curr_max = a[0];

    for (int i = 1; i < size; i++)
    {
        curr_max = max(a[i], curr_max+a[i]);
        max_so_far = max(max_so_far, curr_max);
    }
    return max_so_far;
}

/* Driver program to test maxSubArraySum */
int main()
{
    int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};
    int n = sizeof(a)/sizeof(a[0]);
    int max_sum = maxSubArraySum(a, n);
    cout << "Maximum contiguous sum is " << max_sum;
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to find maximum contiguous subarray

def maxSubArraySum(a,size):

    max_so_far = a[0]
    curr_max = a[0]

    for i in range(1,size):
        curr_max = max(a[i], curr_max + a[i])
        max_so_far = max(max_so_far,curr_max)

    return max_so_far

```

```
# Driver function to check the above function
a = [-2, -3, 4, -1, -2, 1, 5, -3]
print"Maximum contiguous sum is" , maxSubArraySum(a,len(a))
```

#This code is contributed by _Devesh Agrawal_

Run on IDE

Output:

```
Maximum contiguous sum is 7
```

Now try below question

Given an array of integers (possibly some of the elements negative), write a C program to find out the *maximum product* possible by adding 'n' consecutive integers in the array, $n \leq \text{ARRAY_SIZE}$. Also give where in the array this sequence of n integers starts.



References:

http://en.wikipedia.org/wiki/Kadane%27s_Algorithm

Asked in: Amazon, DE Shaw, FactSet, Flipkart, Housing.com, MetLife, Microsoft, Morgan Stanley, Payu, Snapdeal, Teradata, Visa, Walmart, [24]7 Innovation Lab

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Amazon-Question](#) [Dynamic Programming](#) [Visa-Question](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 246 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Longest Palindromic Substring | Set 1

Given a string, find the longest substring which is palindrome. For example, if the given string is "forgeeksskeegfor", the output should be "geeksskeeg".

We strongly recommend that you click here and practice it, before moving on to the solution.

Method 1 (Brute Force)

The simple approach is to check each substring whether the substring is a palindrome or not. We can run three loops, the outer two loops pick all substrings one by one by fixing the corner characters, the inner loop checks whether the picked substring is palindrome or not.

Time complexity: $O(n^3)$

Auxiliary complexity: $O(1)$

Method 2 (Dynamic Programming)

The time complexity can be reduced by storing results of subproblems. The idea is similar to [this post](#). We maintain a boolean table $[n][n]$ that is filled in bottom up manner. The value of $\text{table}[i][j]$ is true, if the substring is palindrome, otherwise false. To calculate $\text{table}[i][j]$, we first check the value of $\text{table}[i+1][j-1]$, if the value is true and $\text{str}[i]$ is same as $\text{str}[j]$, then we make $\text{table}[i][j]$ true. Otherwise, the value of $\text{table}[i][j]$ is made false.

```
// A dynamic programming solution for longest palindr.
// This code is adopted from following link
// http://www.leetcode.com/2011/11/longest-palindromic-substring-part-i.html

#include <stdio.h>
#include <string.h>

// A utility function to print a substring str[low..high]
void printSubStr( char* str, int low, int high )
{
    for( int i = low; i <= high; ++i )
        printf("%c", str[i]);
}

// This function prints the longest palindrome substring
// of str[].
// It also returns the length of the longest palindrome
int longestPalSubstr( char *str )
{
    int n = strlen( str ); // get length of input string

    // table[i][j] will be false if substring str[i..j]
```

```

// is not palindrome.
// Else table[i][j] will be true
bool table[n][n];
memset(table, 0, sizeof(table));

// All substrings of length 1 are palindromes
int maxLength = 1;
for (int i = 0; i < n; ++i)
    table[i][i] = true;

// check for sub-string of length 2.
int start = 0;
for (int i = 0; i < n-1; ++i)
{
    if (str[i] == str[i+1])
    {
        table[i][i+1] = true;
        start = i;
        maxLength = 2;
    }
}

// Check for lengths greater than 2. k is length
// of substring
for (int k = 3; k <= n; ++k)
{
    // Fix the starting index
    for (int i = 0; i < n-k+1 ; ++i)
    {
        // Get the ending index of substring from
        // starting index i and length k
        int j = i + k - 1;

        // checking for sub-string from ith index to
        // jth index iff str[i+1] to str[j-1] is a
        // palindrome
        if (table[i+1][j-1] && str[i] == str[j])
        {
            table[i][j] = true;

            if (k > maxLength)
            {
                start = i;
                maxLength = k;
            }
        }
    }
}

printf("Longest palindrome substring is: ");
printSubStr( str, start, start + maxLength - 1 );

return maxLength; // return length of LPS
}

// Driver program to test above functions
int main()
{
    char str[] = "forgeeksskeegfor";
    printf("\nLength is: %d\n", longestPalSubstr( str ) );
    return 0;
}

```

Run on IDE

Output:

Longest palindrome substring is: geeksskeeg
Length is: 10

Time complexity: O (n²)

Auxiliary Space: O (n²)

We will soon be adding more optimized methods as separate posts.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Strings](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **95** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 23 (Bellman–Ford Algorithm)

Given a graph and a source vertex src in graph, find shortest paths from src to all vertices in the given graph. The graph may contain negative weight edges.

We have discussed [Dijkstra's algorithm](#) for this problem. Dijkstra's algorithm is a Greedy algorithm and time complexity is $O(V \log V)$ (with the use of Fibonacci heap). *Dijkstra doesn't work for Graphs with negative weight edges, Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is $O(VE)$, which is more than Dijkstra.*

Algorithm

Following are the detailed steps.

Input: Graph and a source vertex src

Output: Shortest distance to all vertices from src . If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array $dist[]$ of size $|V|$ with all values as infinite except $dist[src]$ where src is source vertex.

2) This step calculates shortest distances. Do following $|V|-1$ times where $|V|$ is the number of vertices in given graph.

.....**a)** Do following for each edge $u-v$

.....If $dist[v] > dist[u] + \text{weight of edge } uv$, then update $dist[v]$

..... $dist[v] = dist[u] + \text{weight of edge } uv$

3) This step reports if there is a negative weight cycle in graph. Do following for each edge $u-v$

.....If $dist[v] > dist[u] + \text{weight of edge } uv$, then "Graph contains negative weight cycle"

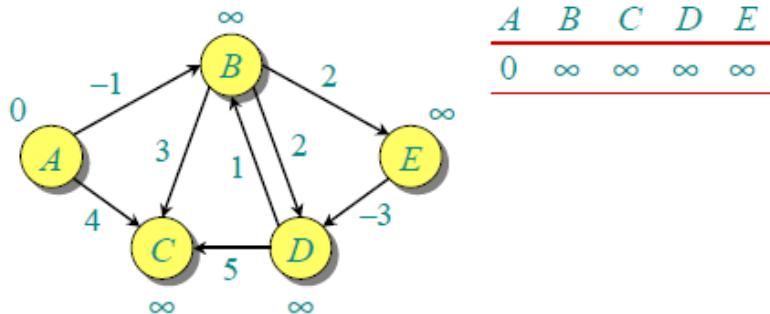
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

How does this work? Like other Dynamic Programming Problems, the algorithm calculate shortest paths in bottom-up manner. It first calculates the shortest distances for the shortest paths which have at-most one edge in the path. Then, it calculates shortest paths with at-most 2 edges, and so on. After the i th iteration of outer loop, the shortest paths with at most i edges are calculated. There can be maximum $|V| - 1$ edges in any simple path, that is why the outer loop runs $|V| - 1$ times. The idea is, assuming that there is no negative weight cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most $(i+1)$ edges (Proof is simple, you can refer [this](#) or [MIT Video Lecture](#)

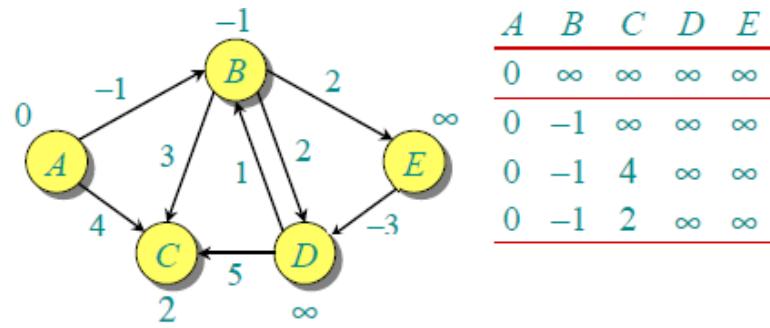
Example

Let us understand the algorithm with following example graph. The images are taken from [this](#) source.

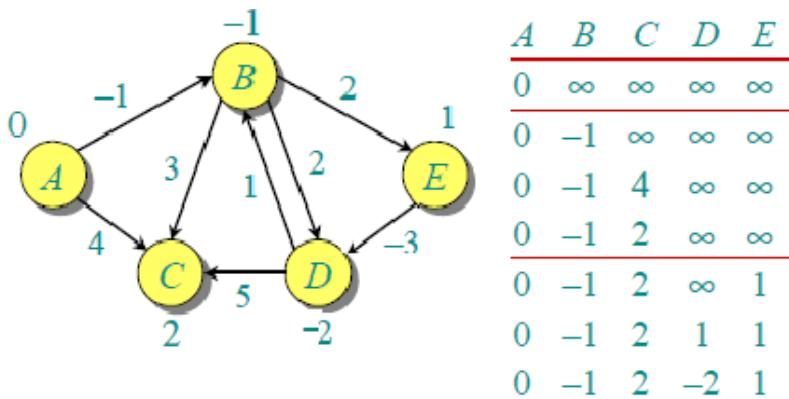
Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so *all edges must be processed 4 times*.



Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.



The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).



The second iteration guarantees to give all shortest paths which are at most 2 edges long. The algorithm processes all edges 2 more times. The distances are minimized after the second iteration, so third and fourth iterations don't update the distances.

Implementation:

```

// A C / C++ program for Bellman-Ford's single source
// shortest path algorithm.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

// a structure to represent a weighted edge in graph
struct Edge
{
    int src, dest, weight;
};

// a structure to represent a connected, directed and
// weighted graph
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges.
    struct Edge* edge;
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph =
        (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge =
        (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

// A utility function used to print the solution
void printArr(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

// The main function that finds shortest distances from src to
// all other vertices using Bellman-Ford algorithm. The function
// also detects negative weight cycle
void BellmanFord(struct Graph* graph, int src)
{
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    // Step 1: Initialize distances from src to all other vertices
    // as INFINITE
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;

    // Step 2: Relax all edges |V| - 1 times. A simple shortest
    // path from src to any other vertex can have at-most |V| - 1
    // edges
    for (int i = 1; i <= V-1; i++)
    {
        for (int j = 0; j < E; j++)
        {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }
}

```

```

    }

    // Step 3: check for negative-weight cycles. The above step
    // guarantees shortest distances if graph doesn't contain
    // negative weight cycle. If we get a shorter path, then there
    // is a cycle.
    for (int i = 0; i < E; i++)
    {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            printf("Graph contains negative weight cycle");
    }

    printArr(dist, V);

    return;
}

// Driver program to test above functions
int main()
{
    /* Let us create the graph given in above example */
    int V = 5; // Number of vertices in graph
    int E = 8; // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1 (or A-B in above figure)
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;

    // add edge 0-2 (or A-C in above figure)
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;

    // add edge 1-2 (or B-C in above figure)
    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;

    // add edge 1-3 (or B-D in above figure)
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;

    // add edge 1-4 (or A-E in above figure)
    graph->edge[4].src = 1;
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;

    // add edge 3-2 (or D-C in above figure)
    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    // add edge 3-1 (or D-B in above figure)
    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;

    // add edge 4-3 (or E-D in above figure)
    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
    graph->edge[7].weight = -3;

    BellmanFord(graph, 0);

    return 0;
}

```

Java

```
// A Java program for Bellman-Ford's single source shortest path
// algorithm.
import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a connected, directed and weighted graph
class Graph
{
    // A class to represent a weighted edge in graph
    class Edge {
        int src, dest, weight;
        Edge() {
            src = dest = weight = 0;
        }
    };

    int V, E;
    Edge edge[];

    // Creates a graph with V vertices and E edges
    Graph(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i=0; i<e; ++i)
            edge[i] = new Edge();
    }

    // The main function that finds shortest distances from src
    // to all other vertices using Bellman-Ford algorithm. The
    // function also detects negative weight cycle
    void BellmanFord(Graph graph,int src)
    {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        // Step 1: Initialize distances from src to all other
        // vertices as INFINITE
        for (int i=0; i<V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;

        // Step 2: Relax all edges |V| - 1 times. A simple
        // shortest path from src to any other vertex can
        // have at-most |V| - 1 edges
        for (int i=1; i<V; ++i)
        {
            for (int j=0; j<E; ++j)
            {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u]!=Integer.MAX_VALUE &&
                    dist[u]+weight<dist[v])
                    dist[v]=dist[u]+weight;
            }
        }

        // Step 3: check for negative-weight cycles. The above
        // step guarantees shortest distances if graph doesn't
        // contain negative weight cycle. If we get a shorter
        // path, then there is a cycle.
        for (int j=0; j<E; ++j)
        {
            int u = graph.edge[j].src;

```

```

        int v = graph.edge[j].dest;
        int weight = graph.edge[j].weight;
        if (dist[u]!=Integer.MAX_VALUE &&
            dist[u]+weight<dist[v])
            System.out.println("Graph contains negative weight cycle");
    }
    printArr(dist, V);
}

// A utility function used to print the solution
void printArr(int dist[], int V)
{
    System.out.println("Vertex   Distance from Source");
    for (int i=0; i<V; ++i)
        System.out.println(i+"\t\t"+dist[i]);
}

// Driver method to test above function
public static void main(String[] args)
{
    int V = 5; // Number of vertices in graph
    int E = 8; // Number of edges in graph

    Graph graph = new Graph(V, E);

    // add edge 0-1 (or A-B in above figure)
    graph.edge[0].src = 0;
    graph.edge[0].dest = 1;
    graph.edge[0].weight = -1;

    // add edge 0-2 (or A-C in above figure)
    graph.edge[1].src = 0;
    graph.edge[1].dest = 2;
    graph.edge[1].weight = 4;

    // add edge 1-2 (or B-C in above figure)
    graph.edge[2].src = 1;
    graph.edge[2].dest = 2;
    graph.edge[2].weight = 3;

    // add edge 1-3 (or B-D in above figure)
    graph.edge[3].src = 1;
    graph.edge[3].dest = 3;
    graph.edge[3].weight = 2;

    // add edge 1-4 (or A-E in above figure)
    graph.edge[4].src = 1;
    graph.edge[4].dest = 4;
    graph.edge[4].weight = 2;

    // add edge 3-2 (or D-C in above figure)
    graph.edge[5].src = 3;
    graph.edge[5].dest = 2;
    graph.edge[5].weight = 5;

    // add edge 3-1 (or D-B in above figure)
    graph.edge[6].src = 3;
    graph.edge[6].dest = 1;
    graph.edge[6].weight = 1;

    // add edge 4-3 (or E-D in above figure)
    graph.edge[7].src = 4;
    graph.edge[7].dest = 3;
    graph.edge[7].weight = -3;

    graph.BellmanFord(graph, 0);
}
}
// Contributed by Aakash Hasija

```

[Run on IDE](#)

Python

```
# Python program for Bellman-Ford's single source
# shortest path algorithm.

from collections import defaultdict

#Class to represent a graph
class Graph:

    def __init__(self,vertices):
        self.V= vertices #No. of vertices
        self.graph = [] # default dictionary to store graph

    # function to add an edge to graph
    def addEdge(self,u,v,w):
        self.graph.append([u, v, w])

    # utility function used to print the solution
    def printArr(self, dist):
        print("Vertex   Distance from Source")
        for i in range(self.V):
            print("%d \t\t %d" % (i, dist[i]))

    # The main function that finds shortest distances from src to
    # all other vertices using Bellman-Ford algorithm. The function
    # also detects negative weight cycle
    def BellmanFord(self, src):

        # Step 1: Initialize distances from src to all other vertices
        # as INFINITE
        dist = [float("Inf")] * self.V
        dist[src] = 0

        # Step 2: Relax all edges |V| - 1 times. A simple shortest
        # path from src to any other vertex can have at-most |V| - 1
        # edges
        for i in range(self.V - 1):
            # Update dist value and parent index of the adjacent vertices of
            # the picked vertex. Consider only those vertices which are still in
            # queue
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w

        # Step 3: check for negative-weight cycles. The above step
        # guarantees shortest distances if graph doesn't contain
        # negative weight cycle. If we get a shorter path, then there
        # is a cycle.

        for u, v, w in self.graph:
            if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                print "Graph contains negative weight cycle"
                return

        # print all distance
        self.printArr(dist)

g = Graph(5)
g.addEdge(0, 1, -1)
g.addEdge(0, 2, 4)
g.addEdge(1, 2, 3)
g.addEdge(1, 3, 2)
g.addEdge(1, 4, 2)
g.addEdge(3, 2, 5)
g.addEdge(3, 1, 1)
g.addEdge(4, 3, -3)

#print the solution
g.BellmanFord(0)
```

Output:

Vertex	Distance from Source
0	0
1	-1
2	2
3	-2
4	1

Notes

- 1) Negative weights are found in various applications of graphs. For example, instead of paying cost for a path, we may get some advantage if we follow the path.
- 2) Bellman-Ford works better (better than Dijksra's) for distributed systems. Unlike Dijksra's where we need to find minimum value of all vertices, in Bellman-Ford, edges are considered one by one.

Exercise

- 1) The standard Bellman-Ford algorithm reports shortest path only if there is no negative weight cycles. Modify it so that it reports minimum distances even if there is a negative weight cycle.
- 2) Can we use Dijksra's algorithm for shortest paths for graphs with negative weights – one idea can be, calculate the minimum weight value, add a positive value (equal to absolute value of minimum weight value) to all weights and run the Dijksra's algorithm for the modified graph. Will this algorithm work?

References:

- <http://www.youtube.com/watch?v=Ttezuzs39nk>
- http://en.wikipedia.org/wiki/Bellman%20%93Ford_algorithm
- <http://www.cs.arizona.edu/classes/cs445/spring07/ShortestPath2.ppt.pdf>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Graph](#) [shortest path](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 35 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 24 (Optimal Binary Search Tree)

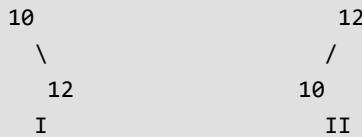
Given a sorted array $keys[0.. n-1]$ of search keys and an array $freq[0.. n-1]$ of frequency counts, where $freq[i]$ is the number of searches to $keys[i]$. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

Example 1

Input: $keys[] = \{10, 12\}$, $freq[] = \{34, 50\}$

There can be following two possible BSTs



Frequency of searches of 10 and 12 are 34 and 50 respectively.

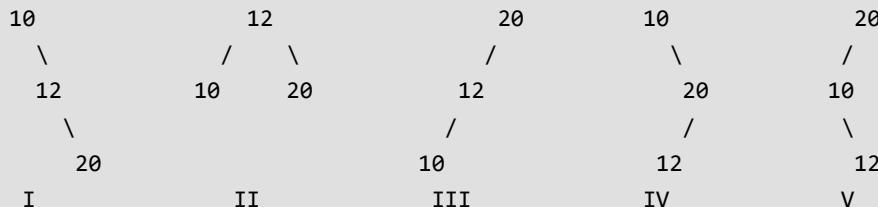
The cost of tree I is $34*1 + 50*2 = 134$

The cost of tree II is $50*1 + 34*2 = 118$

Example 2

Input: $keys[] = \{10, 12, 20\}$, $freq[] = \{34, 8, 50\}$

There can be following possible BSTs



Among all possible BSTs, cost of the fifth BST is minimum.

Cost of the fifth BST is $1*50 + 2*34 + 3*8 = 142$

1) Optimal Substructure:

The optimal cost for $freq[i..j]$ can be recursively calculated using following formula.

$$optCost(i, j) = \sum_{k=i}^j freq[k] + \min_{r=i}^j [optCost(i, r-1) + optCost(r+1, j)]$$

We need to calculate $optCost(0, n-1)$ to find the result.

The idea of above formula is simple, we one by one try all nodes as root (r varies from i to j in second term). When we make r th node as root, we recursively calculate optimal cost from i to r-1 and r+1 to j.

We add sum of frequencies from i to j (see first term in the above formula), this is added because every search will go through root and one comparison will be done for every search.

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

```
// A naive recursive implementation of optimal binary search tree problem
#include <stdio.h>
#include <limits.h>

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j);

// A recursive function to calculate cost of optimal binary search tree
int optCost(int freq[], int i, int j)
{
    // Base cases
    if (j < i)      // If there are no elements in this subarray
        return 0;
    if (j == i)       // If there is one element in this subarray
        return freq[i];

    // Get sum of freq[i], freq[i+1], ... freq[j]
    int fsum = sum(freq, i, j);

    // Initialize minimum value
    int min = INT_MAX;

    // One by one consider all elements as root and recursively find cost
    // of the BST, compare the cost with min and update min if needed
    for (int r = i; r <= j; ++r)
    {
        int cost = optCost(freq, i, r-1) + optCost(freq, r+1, j);
        if (cost < min)
            min = cost;
    }

    // Return minimum value
    return min + fsum;
}

// The main function that calculates minimum cost of a Binary Search Tree.
// It mainly uses optCost() to find the optimal cost.
int optimalSearchTree(int keys[], int freq[], int n)
{
    // Here array keys[] is assumed to be sorted in increasing order.
    // If keys[] is not sorted, then add code to sort keys, and rearrange
    // freq[] accordingly.
    return optCost(freq, 0, n-1);
}

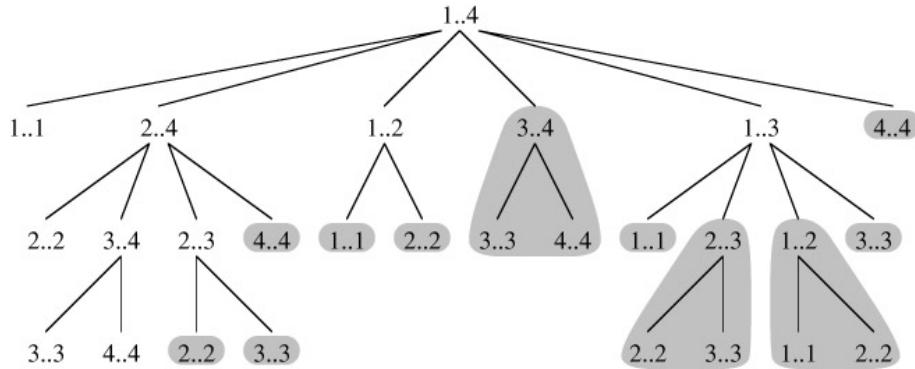
// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <= j; k++)
        s += freq[k];
    return s;
}

// Driver program to test above functions
int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ", optimalSearchTree(keys, freq, n));
    return 0;
}
```

Output:

```
Cost of Optimal BST is 142
```

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. We can see many subproblems being repeated in the following recursion tree for freq[1..4].



Since same subproblems are called again, this problem has Overlapping Subproblems property. So optimal BST problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\)](#) problems, recomputations of same subproblems can be avoided by constructing a temporary array cost[][] in bottom up manner.

Dynamic Programming Solution

Following is C/C++ implementation for optimal BST problem using Dynamic Programming. We use an auxiliary array cost[n][n] to store the solutions of subproblems. cost[0][n-1] will hold the final result. The challenge in implementation is, all diagonal values must be filled first, then the values which lie on the line just above the diagonal. In other words, we must first fill all cost[i][i] values, then all cost[i][i+1] values, then all cost[i][i+2] values. So how to fill the 2D array in such manner? The idea used in the implementation is same as [Matrix Chain Multiplication problem](#), we use a variable 'L' for chain length and increment 'L', one by one. We calculate column number 'j' using the values of 'i' and 'L'.

```
// Dynamic Programming code for Optimal Binary Search Tree Problem
#include <stdio.h>
#include <limits.h>

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j);

/* A Dynamic Programming based function that calculates minimum cost of
   a Binary Search Tree. */
int optimalSearchTree(int keys[], int freq[], int n)
{
    /* Create an auxiliary 2D matrix to store results of subproblems */
    int cost[n][n];

    /* cost[i][j] = Optimal cost of binary search tree that can be
       formed from keys[i] to keys[j].
       cost[0][n-1] will store the resultant cost */

    // For a single key, cost is equal to frequency of the key
    for (int i = 0; i < n; i++)
        cost[i][i] = freq[i];

    // Now we need to consider chains of length 2, 3, ...
    // L is chain length.
    for (int L=2; L<=n; L++)
    {
        for (int i = 0; i < n-L+1; i++)
        {
            int j = i+L-1;
            cost[i][j] = INT_MAX;
            for (int r = i; r <= j; r++)
            {
                int tcost = cost[i][r-1] + cost[r+1][j] + freq[r];
                if (cost[i][j] > tcost)
                    cost[i][j] = tcost;
            }
        }
    }
    return cost[0][n-1];
}
```

```

{
    // i is row number in cost[][]
    for (int i=0; i<=n-L+1; i++)
    {
        // Get column number j from row number i and chain length L
        int j = i+L-1;
        cost[i][j] = INT_MAX;

        // Try making all keys in interval keys[i..j] as root
        for (int r=i; r<=j; r++)
        {
            // c = cost when keys[r] becomes root of this subtree
            int c = ((r > i)? cost[i][r-1]:0) +
                    ((r < j)? cost[r+1][j]:0) +
                    sum(freq, i, j);
            if (c < cost[i][j])
                cost[i][j] = c;
        }
    }
    return cost[0][n-1];
}

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <=j; k++)
        s += freq[k];
    return s;
}

// Driver program to test above functions
int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ", optimalSearchTree(keys, freq, n));
    return 0;
}

```

[Run on IDE](#)

Output:

Cost of Optimal BST is 142

Notes

- 1) The time complexity of the above solution is $O(n^4)$. The time complexity can be easily reduced to $O(n^3)$ by pre-calculating sum of frequencies instead of calling sum() again and again.
- 2) In the above solutions, we have computed optimal cost only. The solutions can be easily modified to store the structure of BSTs also. We can create another auxiliary array of size n to store the structure of tree. All we need to do is, store the chosen 'r' in the innermost loop.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GET 39% OFF
CHEAPEST PRICE FOR THE WORLD'S
MOST TRUSTED SSL

GET YOURS



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.7

Average Difficulty : 3.7/5.0
Based on 34 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 25 (Subset Sum Problem)

Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

```
Examples: set[] = {3, 34, 4, 12, 5, 2}, sum = 9
Output: True //There is a subset (4, 5) with sum 9.
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Let `isSubSetSum(int set[], int n, int sum)` be the function to find whether there is a subset of `set[]` with sum equal to *sum*. *n* is the number of elements in `set[]`.

The `isSubSetSum` problem can be divided into two subproblems

- ...a) Include the last element, recur for $n = n-1$, $sum = sum - set[n-1]$
- ...b) Exclude the last element, recur for $n = n-1$.

If any of the above the above subproblems return true, then return true.

Following is the recursive formula for `isSubSetSum()` problem.

```
isSubSetSum(set, n, sum) = isSubSetSum(set, n-1, sum) ||
                           isSubSetSum(set, n-1, sum-set[n-1])
Base Cases:
isSubSetSum(set, n, sum) = false, if sum > 0 and n == 0
isSubSetSum(set, n, sum) = true, if sum == 0
```

Following is naive recursive implementation that simply follows the recursive structure mentioned above.

```
// A recursive solution for subset sum problem
#include <stdio.h>

// Returns true if there is a subset of set[] with sun equal to given sum
bool isSubSetSum(int set[], int n, int sum)
{
    // Base Cases
    if (sum == 0)
        return true;
```

```

if (n == 0 && sum != 0)
    return false;

// If last element is greater than sum, then ignore it
if (set[n-1] > sum)
    return isSubsetSum(set, n-1, sum);

/* else, check if sum can be obtained by any of the following
   (a) including the last element
   (b) excluding the last element */
return isSubsetSum(set, n-1, sum) ||
        isSubsetSum(set, n-1, sum-set[n-1]);
}

// Driver program to test above function
int main()
{
    int set[] = {3, 34, 4, 12, 5, 2};
    int sum = 9;
    int n = sizeof(set)/sizeof(set[0]);
    if (isSubsetSum(set, n, sum) == true)
        printf("Found a subset with given sum");
    else
        printf("No subset with given sum");
    return 0;
}

```

Run on IDE

Java

```

// A recursive solution for subset sum problem
class subset_sum
{
    // Returns true if there is a subset of set[] with sum
    // equal to given sum
    static boolean isSubsetSum(int set[], int n, int sum)
    {
        // Base Cases
        if (sum == 0)
            return true;
        if (n == 0 && sum != 0)
            return false;

        // If last element is greater than sum, then ignore it
        if (set[n-1] > sum)
            return isSubsetSum(set, n-1, sum);

        /* else, check if sum can be obtained by any of the following
           (a) including the last element
           (b) excluding the last element */
        return isSubsetSum(set, n-1, sum) ||
               isSubsetSum(set, n-1, sum-set[n-1]);
    }
    /* Driver program to test above function */
    public static void main (String args[])
    {
        int set[] = {3, 34, 4, 12, 5, 2};
        int sum = 9;
        int n = set.length;
        if (isSubsetSum(set, n, sum) == true)
            System.out.println("Found a subset with given sum");
        else
            System.out.println("No subset with given sum");
    }
}/* This code is contributed by Rajat Mishra */

```

Run on IDE

Output:

```
Found a subset with given sum
```

The above solution may try all subsets of given set in worst case. Therefore time complexity of the above solution is exponential. The problem is in-fact **NP-Complete** (There is no known polynomial time solution for this problem).

We can solve the problem in Pseudo-polynomial time using Dynamic programming. We create a boolean 2D table `subset[][]` and fill it in bottom up manner. The value of `subset[i][j]` will be true if there is a subset of `set[0..j-1]` with sum equal to `i`, otherwise false. Finally, we return `subset[sum][n]`

```
// A Dynamic Programming solution for subset sum problem
#include <stdio.h>

// Returns true if there is a subset of set[] with sum equal to given sum
bool isSubsetSum(int set[], int n, int sum)
{
    // The value of subset[i][j] will be true if there is a
    // subset of set[0..j-1] with sum equal to i
    bool subset[sum+1][n+1];

    // If sum is 0, then answer is true
    for (int i = 0; i <= n; i++)
        subset[0][i] = true;

    // If sum is not 0 and set is empty, then answer is false
    for (int i = 1; i <= sum; i++)
        subset[i][0] = false;

    // Fill the subset table in bottom up manner
    for (int i = 1; i <= sum; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            subset[i][j] = subset[i][j-1];
            if (i >= set[j-1])
                subset[i][j] = subset[i][j] || subset[i - set[j-1]][j-1];
        }
    }

    /* // uncomment this code to print table
    for (int i = 0; i <= sum; i++)
    {
        for (int j = 0; j <= n; j++)
            printf ("%4d", subset[i][j]);
        printf ("\n");
    } */

    return subset[sum][n];
}

// Driver program to test above function
int main()
{
    int set[] = {3, 34, 4, 12, 5, 2};
    int sum = 9;
    int n = sizeof(set)/sizeof(set[0]);
    if (isSubsetSum(set, n, sum) == true)
        printf("Found a subset with given sum");
    else
        printf("No subset with given sum");
    return 0;
}
```

Run on IDE

Java

```
// A Dynamic Programming solution for subset sum problem
class subset_sum
{
    // Returns true if there is a subset of set[] with sum equal to given sum
    static boolean isSubsetSum(int set[], int n, int sum)
    {
        // The value of subset[i][j] will be true if there
        // is a subset of set[0..j-1] with sum equal to i
        boolean subset[][] = new boolean[sum+1][n+1];

        // If sum is 0, then answer is true
        for (int i = 0; i <= n; i++)
            subset[0][i] = true;

        // If sum is not 0 and set is empty, then answer is false
        for (int i = 1; i <= sum; i++)
            subset[i][0] = false;

        // Fill the subset table in bottom up manner
        for (int i = 1; i <= sum; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                subset[i][j] = subset[i][j-1];
                if (i >= set[j-1])
                    subset[i][j] = subset[i][j] ||
                                  subset[i - set[j-1]][j-1];
            }
        }

        /* // uncomment this code to print table
        for (int i = 0; i <= sum; i++)
        {
            for (int j = 0; j <= n; j++)
                printf ("%4d", subset[i][j]);
            printf ("\n");
        } */

        return subset[sum][n];
    }
    /* Driver program to test above function */
    public static void main (String args[])
    {
        int set[] = {3, 34, 4, 12, 5, 2};
        int sum = 9;
        int n = set.length;
        if (isSubsetSum(set, n, sum) == true)
            System.out.println("Found a subset with given sum");
        else
            System.out.println("No subset with given sum");
    }
}/* This code is contributed by Rajat Mishra */

```

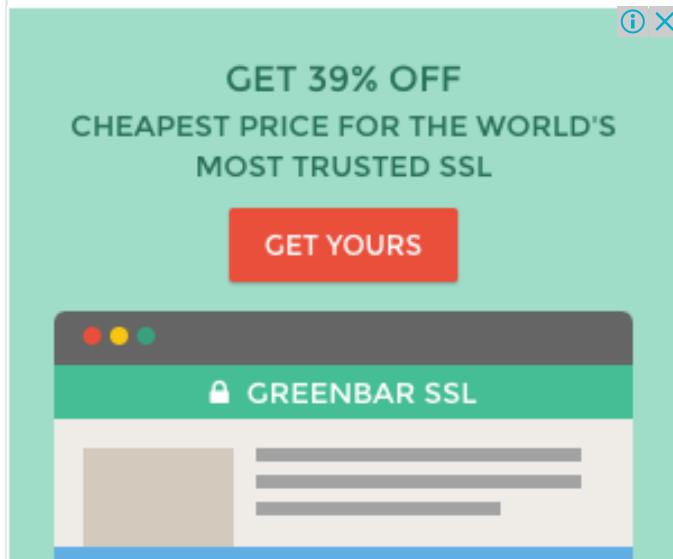
Run on IDE

Output:

Found a subset with given sum

Time complexity of the above solution is O(sum*n).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Dynamic Programming | Adobe-Question | Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.4

Average Difficulty : 3.4/5.0
Based on 73 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search

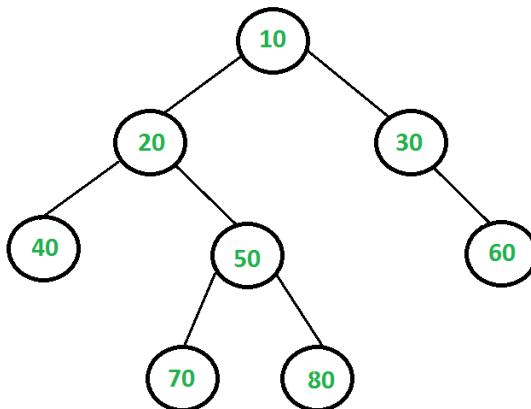


[Login/Register](#)

Dynamic Programming | Set 26 (Largest Independent Set Problem)

Given a Binary Tree, find size of the Largest Independent Set(LIS) in it. A subset of all tree nodes is an independent set if there is no edge between any two nodes of the subset.

For example, consider the following binary tree. The largest independent set(LIS) is {10, 40, 60, 70, 80} and size of the LIS is 5.



A Dynamic Programming solution solves a given problem using solutions of subproblems in bottom up manner. Can the given problem be solved using solutions to subproblems? If yes, then what are the subproblems? Can we find largest independent set size (LISS) for a node X if we know LISS for all descendants of X? If a node is considered as part of LIS, then its children cannot be part of LIS, but its grandchildren can be. Following is optimal substructure property.

1) Optimal Substructure:

Let LISS(X) indicates size of largest independent set of a tree with root X.

$$\text{LISS}(X) = \max \{ (1 + \text{sum of LISS for all grandchildren of } X), (\text{sum of LISS for all children of } X) \}$$

The idea is simple, there are two possibilities for every node X, either X is a member of the set or not a member. If X is a member, then the value of LISS(X) is 1 plus LISS of all grandchildren. If X is not a member, then the value is sum of LISS of all children.

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

```
// A naive recursive implementation of Largest Independent Set problem
#include <stdio.h>
```

```
#include <stdlib.h>

// A utility function to find max of two integers
int max(int x, int y) { return (x > y)? x: y; }

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    struct node *left, *right;
};

// The function returns size of the largest independent set in a given
// binary tree
int LISS(struct node *root)
{
    if (root == NULL)
        return 0;

    // Calculate size excluding the current node
    int size_excl = LISS(root->left) + LISS(root->right);

    // Calculate size including the current node
    int size_incl = 1;
    if (root->left)
        size_incl += LISS(root->left->left) + LISS(root->left->right);
    if (root->right)
        size_incl += LISS(root->right->left) + LISS(root->right->right);

    // Return the maximum of two sizes
    return max(size_incl, size_excl);
}
```

```
// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root      = newNode(20);
    root->left           = newNode(8);
    root->left->left     = newNode(4);
    root->left->right    = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    root->right          = newNode(22);
    root->right->right   = newNode(25);

    printf ("Size of the Largest Independent Set is %d ", LISS(root));

    return 0;
}
```

Run on IDE

Output:

Size of the Largest Independent Set is 5

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. For example, LISS of node with value 50 is evaluated for node

with values 10 and 20 as 50 is grandchild of 10 and child of 20.

Since same subproblems are called again, this problem has Overlapping Subproblems property. So LISS problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

Following is C implementation of Dynamic Programming based solution. In the following solution, an additional field 'liss' is added to tree nodes. The initial value of 'liss' is set as 0 for all nodes. The recursive function LISS() calculates 'liss' for a node only if it is not already set.

```
/* Dynamic programming based program for Largest Independent Set problem */
#include <stdio.h>
#include <stdlib.h>

// A utility function to find max of two integers
int max(int x, int y) { return (x > y)? x: y; }

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    int liss;
    struct node *left, *right;
};

// A memoization function returns size of the largest independent set in
// a given binary tree
int LISS(struct node *root)
{
    if (root == NULL)
        return 0;

    if (root->liss)
        return root->liss;

    if (root->left == NULL && root->right == NULL)
        return (root->liss = 1);

    // Calculate size excluding the current node
    int liss_excl = LISS(root->left) + LISS(root->right);

    // Calculate size including the current node
    int liss_incl = 1;
    if (root->left)
        liss_incl += LISS(root->left->left) + LISS(root->left->right);
    if (root->right)
        liss_incl += LISS(root->right->left) + LISS(root->right->right);

    // Maximum of two sizes is LISS, store it for future uses.
    root->liss = max(liss_incl, liss_excl);

    return root->liss;
}

// A utility function to create a node
struct node* newNode(int data)
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    temp->liss = 0;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root          = newNode(20);
```

```

root->left          = newNode(8);
root->left->left    = newNode(4);
root->left->right   = newNode(12);
root->left->right->left = newNode(10);
root->left->right->right = newNode(14);
root->right         = newNode(22);
root->right->right  = newNode(25);

printf ("Size of the Largest Independent Set is %d ", LISS(root));

return 0;
}

```

[Run on IDE](#)

Output

Size of the Largest Independent Set is 5

Time Complexity: O(n) where n is the number of nodes in given Binary tree.

Following extensions to above solution can be tried as an exercise.

1) Extend the above solution for n-ary tree.

2) The above solution modifies the given tree structure by adding an additional field 'liss' to tree nodes. Extend the solution so that it doesn't modify the tree structure.

3) The above solution only returns size of LIS, it doesn't print elements of LIS. Extend the solution to print all nodes that are part of LIS.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **47** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 27 (Maximum sum rectangle in a 2D matrix)

Given a 2D array, find the maximum sum subarray in it. For example, in the following 2D array, the maximum sum subarray is highlighted with blue rectangle and sum of this subarray is 29.

1	2	-1	-4	-20
-8	-3	4	2	1
3	8	10	1	3
-4	-1	1	7	-6

This problem is mainly an extension of [Largest Sum Contiguous Subarray for 1D array](#).

The **naive solution** for this problem is to check every possible rectangle in given 2D array. This solution requires 4 nested loops and time complexity of this solution would be $O(n^4)$.

Kadane's algorithm for 1D 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 maximum sum contiguous rows for every left and right column pair. We basically find top and bottom row numbers (which have maximum sum) 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. If we apply Kadane's 1D algorithm on temp[], and get the maximum sum subarray of temp, this maximum sum would be the maximum possible sum with left and right as boundary columns. To get the overall maximum sum, we compare this sum with the maximum sum so far.

```
// Program to find maximum sum subarray in a given 2D array
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define ROW 4
#define COL 5

// Implementation of Kadane's algorithm for 1D array. The function
```

```

// returns the maximum sum and stores starting and ending indexes of the
// maximum sum subarray at addresses pointed by start and finish pointers
// respectively.
int kadane(int* arr, int* start, int* finish, int n)
{
    // initialize sum, maxSum and
    int sum = 0, maxSum = INT_MIN, i;

    // Just some initial value to check for all negative values case
    *finish = -1;

    // local variable
    int local_start = 0;

    for (i = 0; i < n; ++i)
    {
        sum += arr[i];
        if (sum < 0)
        {
            sum = 0;
            local_start = i+1;
        }
        else if (sum > maxSum)
        {
            maxSum = sum;
            *start = local_start;
            *finish = i;
        }
    }

    // There is at-least one non-negative number
    if (*finish != -1)
        return maxSum;

    // Special Case: When all numbers in arr[] are negative
    maxSum = arr[0];
    *start = *finish = 0;

    // Find the maximum element in array
    for (i = 1; i < n; i++)
    {
        if (arr[i] > maxSum)
        {
            maxSum = arr[i];
            *start = *finish = i;
        }
    }
    return maxSum;
}

// The main function that finds maximum sum rectangle in M[][][]
void findMaxSum(int M[][COL])
{
    // Variables to store the final output
    int maxSum = INT_MIN, finalLeft, finalRight, finalTop, finalBottom;

    int left, right, i;
    int temp[ROW], sum, start, finish;

    // Set the left column
    for (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 (right = left; right < COL; ++right)
        {
            // Calculate sum between current left and right for every row 'i'
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

            // Find the maximum sum subarray in temp[]. The kadane()
            // function also sets values of start and finish. So 'sum' is
        }
    }
}

```

```

        // sum of rectangle between (start, left) and (finish, right)
        // which is the maximum sum with boundary columns strictly as
        // left and right.
        sum = kadane(temp, &start, &finish, ROW);

        // Compare sum with maximum sum so far. If sum is more, then
        // update maxSum and other output values
        if (sum > maxSum)
        {
            maxSum = sum;
            finalLeft = left;
            finalRight = right;
            finalTop = start;
            finalBottom = finish;
        }
    }

    // Print final values
    printf("(Top, Left) (%d, %d)\n", finalTop, finalLeft);
    printf("(Bottom, Right) (%d, %d)\n", finalBottom, finalRight);
    printf("Max sum is: %d\n", maxSum);
}

// Driver program to test above functions
int main()
{
    int M[ROW][COL] = {{1, 2, -1, -4, -20},
                       {-8, -3, 4, 2, 1},
                       {3, 8, 10, 1, 3},
                       {-4, -1, 1, 7, -6}};
    findMaxSum(M);

    return 0;
}

```

Run on IDE

Java

```

import java.util.*;
import java.lang.*;
import java.io.*;

/**
 * Given a 2D array, find the maximum sum subarray in it
 */
class Ideone
{
    public static void main (String[] args) throws java.lang.Exception
    {
        findMaxSubMatrix(new int[][] {
            {1, 2, -1, -4, -20},
            {-8, -3, 4, 2, 1},
            {3, 8, 10, 1, 3},
            {-4, -1, 1, 7, -6}
        });
    }

    /**
     * To find maxSum in 1d array
     *
     * return {maxSum, left, right}
     */
    public static int[] kadane(int[] a) {
        //result[0] == maxSum, result[1] == start, result[2] == end;
        int[] result = new int[]{Integer.MIN_VALUE, 0, -1};
        int currentSum = 0;
        int localStart = 0;

```

```

    for (int i = 0; i < a.length; i++) {
        currentSum += a[i];
        if (currentSum < 0) {
            currentSum = 0;
            localStart = i + 1;
        } else if (currentSum > result[0]) {
            result[0] = currentSum;
            result[1] = localStart;
            result[2] = i;
        }
    }

    //all numbers in a are negative
    if (result[2] == -1) {
        result[0] = 0;
        for (int i = 0; i < a.length; i++) {
            if (a[i] > result[0]) {
                result[0] = a[i];
                result[1] = i;
                result[2] = i;
            }
        }
    }

    return result;
}

/**
 * To find and print maxSum, (left, top),(right, bottom)
 */
public static void findMaxSubMatrix(int[][] a) {
    int cols = a[0].length;
    int rows = a.length;
    int[] currentResult;
    int maxSum = Integer.MIN_VALUE;
    int left = 0;
    int top = 0;
    int right = 0;
    int bottom = 0;

    for (int leftCol = 0; leftCol < cols; leftCol++) {
        int[] tmp = new int[rows];

        for (int rightCol = leftCol; rightCol < cols; rightCol++) {

            for (int i = 0; i < rows; i++) {
                tmp[i] += a[i][rightCol];
            }
            currentResult = kadane(tmp);
            if (currentResult[0] > maxSum) {
                maxSum = currentResult[0];
                left = leftCol;
                top = currentResult[1];
                right = rightCol;
                bottom = currentResult[2];
            }
        }
    }
    System.out.println("MaxSum: " + maxSum +
        ", range: [(" + left + ", " + top +
        ") (" + right + ", " + bottom + ")]");
}
// Thanks to Ilia Savin for contributing this code.

```

[Run on IDE](#)

Output:

(Top, Left) (1, 1)
 (Bottom, Right) (3, 3)
 Max sum is: 29

Time Complexity: O(n^3)

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



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Matrix](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.4

Average Difficulty : 4.4/5.0
 Based on 79 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count number of binary strings without consecutive 1's

Given a positive integer N, count all possible distinct binary strings of length N such that there are no consecutive 1's.

Examples:

```
Input: N = 2
Output: 3
// The 3 strings are 00, 01, 10
```

```
Input: N = 3
Output: 5
// The 5 strings are 000, 001, 010, 100, 101
```

We strongly recommend that you click here and practice it, before moving on to the solution.

This problem can be solved using Dynamic Programming. Let $a[i]$ be the number of binary strings of length i which do not contain any two consecutive 1's and which end in 0. Similarly, let $b[i]$ be the number of such strings which end in 1. We can append either 0 or 1 to a string ending in 0, but we can only append 0 to a string ending in 1. This yields the recurrence relation:

```
a[i] = a[i - 1] + b[i - 1]
b[i] = a[i - 1]
```

The base cases of above recurrence are $a[1] = b[1] = 1$. The total number of strings of length i is just $a[i] + b[i]$.

Following is the implementation of above solution. In the following implementation, indexes start from 0. So $a[i]$ represents the number of binary strings for input length $i+1$. Similarly, $b[i]$ represents binary strings for input length $i+1$.

```
// C++ program to count all distinct binary strings
// without two consecutive 1's
#include <iostream>
using namespace std;
```

```

int countStrings(int n)
{
    int a[n], b[n];
    a[0] = b[0] = 1;
    for (int i = 1; i < n; i++)
    {
        a[i] = a[i-1] + b[i-1];
        b[i] = a[i-1];
    }
    return a[n-1] + b[n-1];
}

// Driver program to test above functions
int main()
{
    cout << countStrings(3) << endl;
    return 0;
}

```

[Run on IDE](#)

Java

```

class Subset_sum
{
    static int countStrings(int n)
    {
        int a[] = new int [n];
        int b[] = new int [n];
        a[0] = b[0] = 1;
        for (int i = 1; i < n; i++)
        {
            a[i] = a[i-1] + b[i-1];
            b[i] = a[i-1];
        }
        return a[n-1] + b[n-1];
    }
    /* Driver program to test above function */
    public static void main (String args[])
    {
        System.out.println(countStrings(3));
    }
}/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

5

Source:

courses.csail.mit.edu/6.006/oldquizzes/solutions/q2-f2009-sol.pdf

If we take a closer look at the pattern, we can observe that the count is actually $(n+2)$ 'th Fibonacci number for $n \geq 1$. The **Fibonacci Numbers** are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 141,

```

n = 1, count = 2  = fib(3)
n = 2, count = 3  = fib(4)
n = 3, count = 5  = fib(5)
n = 4, count = 8  = fib(6)
n = 5, count = 13 = fib(7)
.....

```

Therefore we can count the strings in O(Log n) time also using the method 5 [here](#).

Asked in: [Flipkart](#), [Microsoft](#), [Snapdeal](#)

This article is contributed by [Rahul Jain](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Free Download

Zip, Unzip or Open Any File!

Unzipper



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [binary-string](#) [Dynamic Programming](#) [Fibonacci](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

2.9

Average Difficulty : 2.9/5.0
Based on 50 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Dynamic Programming | Set 37 (Boolean Parenthesization Problem)

Given a boolean expression with following symbols.

Symbols

```
'T' ---> true
'F' ---> false
```

And following operators filled between symbols

Operators

```
& ---> boolean AND
| ---> boolean OR
^ ---> boolean XOR
```

Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.

Let the input be in form of two arrays one contains the symbols (T and F) in order and other contains operators (&, | and ^)

Examples:

```
Input: symbol[] = {T, F, T}
       operator[] = {^, &}
```

Output: 2

The given expression is "T ^ F & T", it evaluates true
in two ways "((T ^ F) & T)" and "(T ^ (F & T))"

```
Input: symbol[] = {T, F, F}
       operator[] = {^, |}
```

Output: 2

The given expression is "T ^ F | F", it evaluates true
in two ways "((T ^ F) | F)" and "(T ^ (F | F))".

```
Input: symbol[] = {T, T, F, T}
       operator[] = {|, &, ^}
```

Output: 4

The given expression is "T | T & F ^ T", it evaluates true
in 4 ways ((T|T)&(F^T)), (T|(T&(F^T))), (((T|T)&F)^T)
and (T|((T&F)^T)).

Solution:

Let $T(i, j)$ represents the number of ways to parenthesize the symbols between i and j (both inclusive) such that the subexpression between i and j evaluates to true.

$$T(i, j) = \sum_{k=i}^{j-1} \begin{cases} T(i, k) * T(k+1, j) & \text{If operator}[k] \text{ is } \& \\ Total(i, k) * Total(k+1, j) - F(i, k) * F(k+1, j) & \text{If operator}[k] \text{ is } | \\ T(i, k) * F(k+1, j) + F(i, k) * T(k+1) & \text{If operator}[k] \text{ is } ^\wedge \end{cases}$$

$$\text{Total}(i, j) = T(i, j) + F(i, j)$$

Let $F(i, j)$ represents the number of ways to parenthesize the symbols between i and j (both inclusive) such that the subexpression between i and j evaluates to false.

$$F(i, j) = \sum_{k=i}^{j-1} \begin{cases} Total(i, k) * Total(k+1, j) - T(i, k) * T(k+1, j) & \text{If operator}[k] \text{ is } \& \\ F(i, k) * F(k+1, j) & \text{If operator}[k] \text{ is } | \\ T(i, k) * T(k+1, j) + F(i, k) * F(k+1) & \text{If operator}[k] \text{ is } ^\wedge \end{cases}$$

$$\text{Total}(i, j) = T(i, j) + F(i, j)$$

Base Cases:

```
T(i, i) = 1 if symbol[i] = 'T'
T(i, i) = 0 if symbol[i] = 'F'
```

```
F(i, i) = 1 if symbol[i] = 'F'
F(i, i) = 0 if symbol[i] = 'T'
```

If we draw recursion tree of above recursive solution, we can observe that it many overlapping subproblems. Like other **dynamic programming problems**, it can be solved by filling a table in bottom up manner. Following is C++ implementation of dynamic programming solution.

```
#include<iostream>
#include<cstring>
using namespace std;

// Returns count of all possible parenthesizations that lead to
// result true for a boolean expression with symbols like true
// and false and operators like &, | and ^ filled between symbols
int countParenth(char symb[], char oper[], int n)
{
    int F[n][n], T[n][n];

    // Fill diagonal entries first
    // All diagonal entries in T[i][i] are 1 if symbol[i]
    // is T (true). Similarly, all F[i][i] entries are 1 if
    // symbol[i] is F (False)
    for (int i = 0; i < n; i++)
    {
        F[i][i] = (symb[i] == 'F')? 1: 0;
        T[i][i] = (symb[i] == 'T')? 1: 0;
    }

    // Now fill T[i][i+1], T[i][i+2], T[i][i+3]... in order
    // And F[i][i+1], F[i][i+2], F[i][i+3]... in order
    for (int gap=1; gap<n; ++gap)
    {
        for (int i=0, j=gap; j<n; ++i, ++j)
        {
            T[i][j] = F[i][j] = 0;
            for (int g=0; g<gap; g++)
            {
                // Find place of parenthesization using current value
                // of gap
                int k = i + g;

                // Store Total[i][k] and Total[k+1][j]
            }
        }
    }
}
```

```

int tik = T[i][k] + F[i][k];
int tkj = T[k+1][j] + F[k+1][j];

// Follow the recursive formulas according to the current
// operator
if (oper[k] == '&')
{
    T[i][j] += T[i][k]*T[k+1][j];
    F[i][j] += (tik*tkj - T[i][k]*T[k+1][j]);
}
if (oper[k] == '|')
{
    F[i][j] += F[i][k]*F[k+1][j];
    T[i][j] += (tik*tkj - F[i][k]*F[k+1][j]);
}
if (oper[k] == '^')
{
    T[i][j] += F[i][k]*T[k+1][j] + T[i][k]*F[k+1][j];
    F[i][j] += T[i][k]*T[k+1][j] + F[i][k]*F[k+1][j];
}

return T[0][n-1];
}

// Driver program to test above function
int main()
{
    char symbols[] = "TTFT";
    char operators[] = "|&^";
    int n = strlen(symbols);

    // There are 4 ways
    // ((T|T)&(F^T)), (T|(T&(F^T))), (((T|T)&F)^T) and (T|((T&F)^T))
    cout << countParenth(symbols, operators, n);
    return 0;
}

```

[Run on IDE](#)

Output:

4

Time Complexity: $O(n^3)$

Auxiliary Space: $O(n^2)$

References:

http://people.cs.clemson.edu/~bcdean/dp_practice/dp_9.swf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Live There with Airbnb

From bedrooms to
apartments to villas -
Airbnb has it all.

airbnb.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.5

Average Difficulty : 4.5/5.0
Based on 67 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

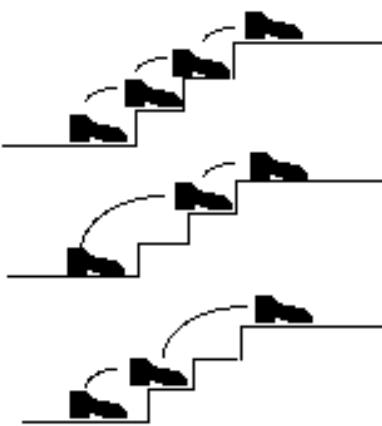
[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Count ways to reach the n'th stair

There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1 stair or 2 stairs at a time. Count the number of ways, the person can reach the top.

Consider the example shown in diagram. The value of n is 3. There are 3 ways to reach the top. The diagram is taken from [Easier Fibonacci puzzles](#)



More Examples:

Input: $n = 1$

Output: 1

There is only one way to climb 1 stair

Input: $n = 2$

Output: 2

There are two ways: (1, 1) and (2)

Input: $n = 4$

Output: 5

(1, 1, 1, 1), (1, 1, 2), (2, 1, 1), (1, 2, 1), (2, 2)

We strongly recommend that you click here and practice it, before moving on to the solution.

We can easily find recursive nature in above problem. The person can reach n 'th stair from either $(n-1)$ 'th stair or from $(n-2)$ 'th stair. Let the total number of ways to reach n 't stair be ' $\text{ways}(n)$ '. The value of ' $\text{ways}(n)$ ' can be written as following.

```
ways(n) = ways(n-1) + ways(n-2)
```

The above expression is actually the expression for **Fibonacci numbers**, but there is one thing to notice, the value of ways(n) is equal to fibonacci(n+1).

```
ways(1) = fib(2) = 1
ways(2) = fib(3) = 2
ways(3) = fib(4) = 3
```

So we can use function for fibonacci numbers to find the value of ways(n). Following is C++ implementation of the above idea.

```
// A C program to count number of ways to reach n'th stair when
// a person can climb 1, 2, ...m stairs at a time.
#include<stdio.h>

// A simple recursive program to find n'th fibonacci number
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}

// Returns number of ways to reach s'th stair
int countWays(int s)
{
    return fib(s + 1);
}

// Driver program to test above functions
int main ()
{
    int s = 4;
    printf("Number of ways = %d", countWays(s));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Java

```
class stairs
{
    // A simple recursive program to find n'th fibonacci number
    static int fib(int n)
    {
        if (n <= 1)
            return n;
        return fib(n-1) + fib(n-2);
    }

    // Returns number of ways to reach s'th stair
    static int countWays(int s)
    {
        return fib(s + 1);
    }

    /* Driver program to test above function */
    public static void main (String args[])
}
```

```

{
    int s = 4;
    System.out.println("Number of ways = "+ countWays(s));
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

Number of ways = 5

The time complexity of the above implementation is exponential (golden ratio raised to power n). It can be optimized to work in O(Logn) time using the previously [discussed Fibonacci function optimizations](#).

Generalization of the above problem

How to count number of ways if the person can climb up to m stairs for a given value m? For example if m is 4, the person can climb 1 stair or 2 stairs or 3 stairs or 4 stairs at a time.

We can write the recurrence as following.

$$\text{ways}(n, m) = \text{ways}(n-1, m) + \text{ways}(n-2, m) + \dots + \text{ways}(n-m, m)$$

Following is C++ implementation of above recurrence.

```

// A C program to count number of ways to reach n'th stair when
// a person can climb either 1 or 2 stairs at a time
#include<stdio.h>

// A recursive function used by countWays
int countWaysUtil(int n, int m)
{
    if (n <= 1)
        return n;
    int res = 0;
    for (int i = 1; i<=m && i<=n; i++)
        res += countWaysUtil(n-i, m);
    return res;
}

// Returns number of ways to reach s'th stair
int countWays(int s, int m)
{
    return countWaysUtil(s+1, m);
}

// Driver program to test above functions
int main ()
{
    int s = 4, m = 2;
    printf("Number of ways = %d", countWays(s, m));
    return 0;
}

```

[Run on IDE](#)

Java

```

class stairs
{
    // A recursive function used by countWays
    static int countWaysUtil(int n, int m)
    {
        if (n <= 1)
            return n;
        int res = 0;
        for (int i = 1; i<=m && i<=n; i++)
            res += countWaysUtil(n-i, m);
        return res;
    }

    // Returns number of ways to reach s'th stair
    static int countWays(int s, int m)
    {
        return countWaysUtil(s+1, m);
    }
}

/* Driver program to test above function */
public static void main (String args[])
{
    int s = 4, m = 2;
    System.out.println("Number of ways = "+ countWays(s,m));
}
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Output:

Number of ways = 5

The time complexity of above solution is exponential. It can be optimized to O(mn) by using dynamic programming. Following is dynamic programming based solution. We build a table res[] in bottom up manner.

```

// A C program to count number of ways to reach n'th stair when
// a person can climb 1, 2, ...m stairs at a time
#include<stdio.h>

// A recursive function used by countWays
int countWaysUtil(int n, int m)
{
    int res[n];
    res[0] = 1; res[1] = 1;
    for (int i=2; i<n; i++)
    {
        res[i] = 0;
        for (int j=1; j<=m && j<=i; j++)
            res[i] += res[i-j];
    }
    return res[n-1];
}

// Returns number of ways to reach s'th stair
int countWays(int s, int m)
{
    return countWaysUtil(s+1, m);
}

// Driver program to test above functions
int main ()
{
    int s = 4, m = 2;

```

```

printf("Nuber of ways = %d", countWays(s, m));
return 0;
}

```

[Run on IDE](#)

Java

```

class stairs
{
    // A recursive function used by countWays
    static int countWaysUtil(int n, int m)
    {
        if (n <= 1)
            return n;
        int res = 0;
        for (int i = 1; i<=m && i<=n; i++)
            res += countWaysUtil(n-i, m);
        return res;
    }
    // Returns number of ways to reach s'th stair
    static int countWays(int s, int m)
    {
        return countWaysUtil(s+1, m);
    }

    /* Driver program to test above function */
    public static void main (String args[])
    {
        int s = 4,m = 2;
        System.out.println("Number of ways = "+ countWays(s,m));
    }
}/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

Number of ways = 5

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



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Mathematical

Related Posts:

- Nth character in Concatenated Decimal String
- Find the highest occurring digit in prime numbers in a range
- Smallest number to multiply to convert floating point to natural
- Generate all palindromic numbers less than n
- Number of sextuplets (or six values) that satisfy an equation
- Circular primes less than n
- Sphenic Number
- Minimum sum of two numbers formed from digits of an array

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 49 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



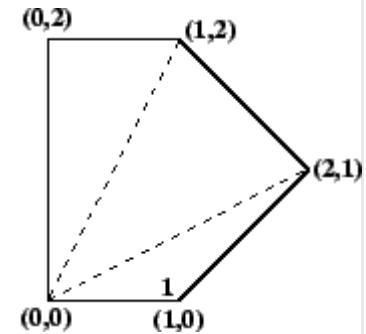
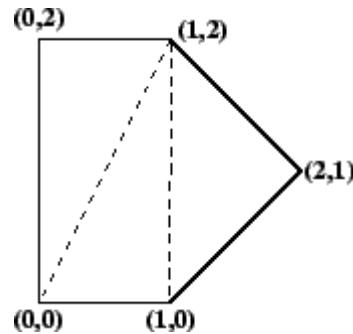
[Login/Register](#)

Minimum Cost Polygon Triangulation

A triangulation of a convex polygon is formed by drawing diagonals between non-adjacent vertices (corners) such that the diagonals never intersect. The problem is to find the cost of triangulation with the minimum cost. The cost of a triangulation is sum of the weights of its component triangles. Weight of each triangle is its perimeter (sum of lengths of all sides)

See following example taken from [this](#) source.

Two triangulations of the same convex pentagon. The triangulation on the left has a cost of $8 + 2\sqrt{2} + 2\sqrt{5}$ (approximately 15.30), the one on the right has a cost of $4 + 2\sqrt{2} + 4\sqrt{5}$ (approximately 15.77).



This problem has recursive substructure. The idea is to divide the polygon into three parts: a single triangle, the sub-polygon to the left, and the sub-polygon to the right. We try all possible divisions like this and find the one that minimizes the cost of the triangle plus the cost of the triangulation of the two sub-polygons.

```
Let Minimum Cost of triangulation of vertices from i to j be minCost(i, j)
If j <= i + 2 Then
    minCost(i, j) = 0
Else
    minCost(i, j) = Min { minCost(i, k) + minCost(k, j) + cost(i, k, j) }
        Here k varies from 'i+1' to 'j-1'
```

Cost of a triangle formed by edges (i, j), (j, k) and (k, i) is

$$\text{cost}(i, j, k) = \text{dist}(i, j) + \text{dist}(j, k) + \text{dist}(k, i)$$

Following is C++ implementation of above naive recursive formula.

```
// Recursive implementation for minimum cost convex polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Function to calculate distance between two points
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) +
                (p1.y - p2.y) * (p1.y - p2.y));
}

// Function to calculate cost of a triangle with vertices i, j, k
double cost(Point p1, Point p2, Point p3)
{
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// Function to calculate minimum cost of triangulation of a polygon with n vertices
double minCost(int i, int j)
{
    if (j <= i + 2)
        return 0;

    double min_val = MAX;
    for (int k = i + 1; k < j; k++)
        min_val = min(min_val, minCost(i, k) + minCost(k, j) + cost(p[i], p[k], p[j]));
    return min_val;
}
```

```

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A recursive function to find minimum cost of polygon triangulation
// The polygon is represented by points[i..j].
double mTC(Point points[], int i, int j)
{
    // There must be at least three points between i and j
    // (including i and j)
    if (j < i+2)
        return 0;

    // Initialize result as infinite
    double res = MAX;

    // Find minimum triangulation by considering all
    for (int k=i+1; k<j; k++)
        res = min(res, (mTC(points, i, k) + mTC(points, k, j) +
                        cost(points, i, k, j)));
    return res;
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTC(points, 0, n-1);
    return 0;
}

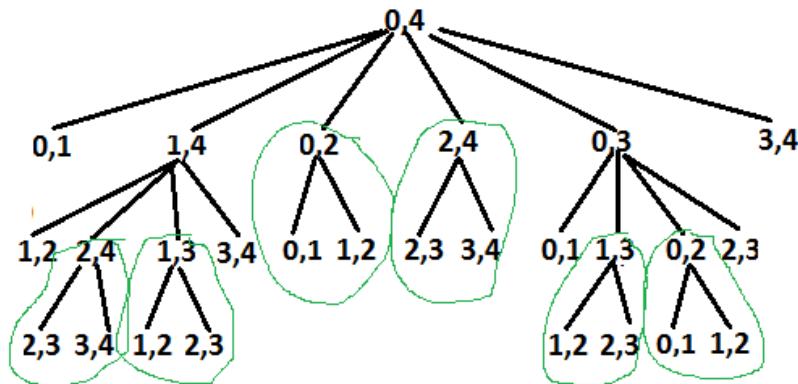
```

Run on IDE

Output:

15.3006

The above problem is similar to [Matrix Chain Multiplication](#). The following is recursion tree for $mTC(\text{points}[0..4])$.



Recursion Tree for recursive implementation. Overlapping subproblems are encircled.

It can be easily seen in the above recursion tree that the problem has many overlapping subproblems. Since the problem has both properties: **Optimal Substructure** and **Overlapping Subproblems**, it can be efficiently solved using dynamic programming.

Following is C++ implementation of dynamic programming solution.

```

// A Dynamic Programming based program to find minimum cost of convex
// polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A Dynamic programming based function to find minimum cost for convex
// polygon triangulation.
double mTCDP(Point points[], int n)
{
    // There must be at least 3 points to form a triangle
    if (n < 3)
        return 0;

    // table to store results of subproblems. table[i][j] stores cost of
    // triangulation of points from i to j. The entry table[0][n-1] stores
    // the final result.
    double table[n][n];
  
```

```

// Fill table using above recursive formula. Note that the table
// is filled in diagonal fashion i.e., from diagonal elements to
// table[0][n-1] which is the result.
for (int gap = 0; gap < n; gap++)
{
    for (int i = 0, j = gap; j < n; i++, j++)
    {
        if (j < i+2)
            table[i][j] = 0.0;
        else
        {
            table[i][j] = MAX;
            for (int k = i+1; k < j; k++)
            {
                double val = table[i][k] + table[k][j] + cost(points,i,j,k);
                if (table[i][j] > val)
                    table[i][j] = val;
            }
        }
    }
}
return table[0][n-1];
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTCDP(points, n);
    return 0;
}

```

[Run on IDE](#)

Output:

15.3006

Time complexity of the above dynamic programming solution is $O(n^3)$.

Please note that the above implementations assume that the points of convex polygon are given in order (either clockwise or anticlockwise)

Exercise:

Extend the above solution to print triangulation also. For the above example, the optimal triangulation is 0 3 4, 0 1 3, and 1 2 3.

Sources:

<http://www.cs.utexas.edu/users/djimenez/utsa/cs3343/lecture12.html>

<http://www.cs.utoronto.ca/~heap/Courses/270F02/A4/chains/node2.html>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Geometric Dynamic Programming geometric algorithms

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.4

Average Difficulty : 4.4/5.0
Based on 28 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Mobile Numeric Keypad Problem

Given the mobile numeric keypad. You can only press buttons that are up, left, right or down to the current button. You are not allowed to press bottom row corner buttons (i.e. * and #).

Given a number N, find out the number of possible numbers of given length.

Examples:

For N=1, number of possible numbers would be 10 (0, 1, 2, 3,, 9)

For N=2, number of possible numbers would be 36

Possible numbers: 00,08 11,12,14 22,21,23,25 and so on.

If we start with 0, valid numbers will be 00, 08 (count: 2)

If we start with 1, valid numbers will be 11, 12, 14 (count: 3)

If we start with 2, valid numbers will be 22, 21, 23,25 (count: 4)

If we start with 3, valid numbers will be 33, 32, 36 (count: 3)

If we start with 4, valid numbers will be 44,41,45,47 (count: 4)

If we start with 5, valid numbers will be 55,54,52,56,58 (count: 5)

1	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUV	9 WXYZ
	0	*

We need to print the count of possible numbers.

We strongly recommend to minimize the browser and try this yourself first.

N = 1 is trivial case, number of possible numbers would be 10 (0, 1, 2, 3,, 9)

For N > 1, we need to start from some button, then move to any of the four direction (up, left, right or down) which takes to a valid button (should not go to *, #). Keep doing this until N length number is obtained (depth first traversal).

Recursive Solution:

Mobile Keypad is a rectangular grid of 4X3 (4 rows and 3 columns)

Lets say Count(i, j, N) represents the count of N length numbers starting from position (i, j)

```

If N = 1
    Count(i, j, N) = 10
Else
    Count(i, j, N) = Sum of all Count(r, c, N-1) where (r, c) is new
                    position after valid move of length 1 from current
                    position (i, j)
  
```

Following is C implementation of above recursive formula.

```
// A Naive Recursive C program to count number of possible numbers
// of given length
#include <stdio.h>

// left, up, right, down move from current location
int row[] = {0, 0, -1, 0, 1};
int col[] = {0, -1, 0, 1, 0};

// Returns count of numbers of length n starting from key position
// (i, j) in a numeric keyboard.
int getCountUtil(char keypad[][3], int i, int j, int n)
{
    if (keypad == NULL || n <= 0)
        return 0;

    // From a given key, only one number is possible of length 1
    if (n == 1)
        return 1;

    int k=0, move=0, ro=0, co=0, totalCount = 0;

    // move left, up, right, down from current location and if
    // new location is valid, then get number count of length
    // (n-1) from that new position and add in count obtained so far
    for (move=0; move<5; move++)
    {
        ro = i + row[move];
        co = j + col[move];
        if (ro >= 0 && ro <= 3 && co >=0 && co <= 2 &&
            keypad[ro][co] != '*' && keypad[ro][co] != '#')
        {
            totalCount += getCountUtil(keypad, ro, co, n-1);
        }
    }

    return totalCount;
}

// Return count of all possible numbers of length n
// in a given numeric keyboard
int getCount(char keypad[][3], int n)
{
    // Base cases
    if (keypad == NULL || n <= 0)
        return 0;
    if (n == 1)
        return 10;

    int i=0, j=0, totalCount = 0;
    for (i=0; i<4; i++) // Loop on keypad row
    {
        for (j=0; j<3; j++) // Loop on keypad column
        {
            // Process for 0 to 9 digits
            if (keypad[i][j] != '*' && keypad[i][j] != '#')
            {
                // Get count when number is starting from key
                // position (i, j) and add in count obtained so far
                totalCount += getCountUtil(keypad, i, j, n);
            }
        }
    }
    return totalCount;
}

// Driver program to test above function
int main(int argc, char *argv[])
{
    char keypad[4][3] = {{'1','2','3'},
                        {'4','5','6'},
                        {'7','8','9'},
                        {'*','0','#'}};
}
```

```

printf("Count for numbers of length %d: %d\n", 1, getCount(keypad, 1));
printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));
printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));
printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));
printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));

return 0;
}

```

[Run on IDE](#)

Output:

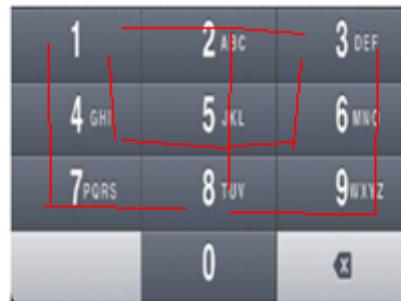
```

Count for numbers of length 1: 10
Count for numbers of length 2: 36
Count for numbers of length 3: 138
Count for numbers of length 4: 532
Count for numbers of length 5: 2062

```

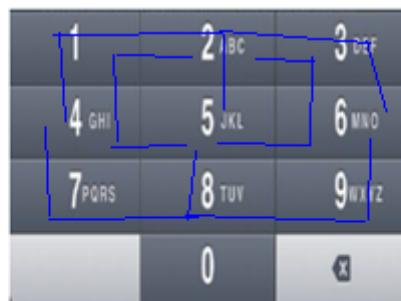
Dynamic Programming

There are many repeated traversal on smaller paths (traversal for smaller N) to find all possible longer paths (traversal for bigger N). See following two diagrams for example. In this traversal, for N = 4 from two starting positions (buttons '4' and '8'), we can see there are few repeated traversals for N = 2 (e.g. 4 -> 1, 6 -> 3, 8 -> 9, 8 -> 7 etc).



Few traversals starting for button 8 for N = 4

e.g. 8 -> 7 -> 4 -> 1, 8 -> 9 -> 6 -> 3
 8 -> 5 -> 4 -> 1, 8 -> 5 -> 6 -> 3
 8 -> 5 -> 2 -> 2, 8 -> 5 -> 2 -> 3



Few traversals starting from button 5 for N=4

e.g. 5 -> 8 -> 7 -> 4, 5 -> 8 -> 9 -> 6
 5 -> 4 -> 1 -> 2, 5 -> 6 -> 3 -> 2
 5 -> 2 -> 1 -> 4, 5 -> 2 -> 3 -> 6

Since the problem has both properties: **Optimal Substructure** and **Overlapping Subproblems**, it can be efficiently solved using dynamic programming.

Following is C program for dynamic programming implementation.

```
// A Dynamic Programming based C program to count number of
// possible numbers of given length
#include <stdio.h>

// Return count of all possible numbers of length n
// in a given numeric keyboard
int getCount(char keypad[][3], int n)
{
    if(keypad == NULL || n <= 0)
        return 0;
    if(n == 1)
        return 10;

    // left, up, right, down move from current location
    int row[] = {0, 0, -1, 0, 1};
    int col[] = {0, -1, 0, 1, 0};

    // taking n+1 for simplicity - count[i][j] will store
    // number count starting with digit i and length j
    int count[10][n+1];
    int i=0, j=0, k=0, move=0, ro=0, co=0, num = 0;
    int nextNum=0, totalCount = 0;

    // count numbers starting with digit i and of lengths 0 and 1
    for (i=0; i<=9; i++)
    {
        count[i][0] = 0;
        count[i][1] = 1;
    }

    // Bottom up - Get number count of length 2, 3, 4, ... , n
    for (k=2; k<=n; k++)
    {
        for (i=0; i<4; i++) // Loop on keypad row
        {
            for (j=0; j<3; j++) // Loop on keypad column
            {
                // Process for 0 to 9 digits
                if (keypad[i][j] != '*' && keypad[i][j] != '#')
                {
                    // Here we are counting the numbers starting with
                    // digit keypad[i][j] and of length k keypad[i][j]
                    // will become 1st digit, and we need to look for
                    // (k-1) more digits
                    num = keypad[i][j] - '0';
                    count[num][k] = 0;

                    // move left, up, right, down from current location
                    // and if new location is valid, then get number
                    // count of length (k-1) from that new digit and
                    // add in count we found so far
                    for (move=0; move<5; move++)
                    {
                        ro = i + row[move];
                        co = j + col[move];
                        if (ro >= 0 && ro <= 3 && co >=0 && co <= 2 &&
                            keypad[ro][co] != '*' && keypad[ro][co] != '#')
                        {
                            nextNum = keypad[ro][co] - '0';
                            count[num][k] += count[nextNum][k-1];
                        }
                    }
                }
            }
        }
    }

    // Get count of all possible numbers of length "n" starting
    // with digit 0, 1, 2, ..., 9
    totalCount = 0;
    for (i=0; i<=9; i++)
        totalCount += count[i][n];
}
```

```

    return totalCount;
}

// Driver program to test above function
int main(int argc, char *argv[])
{
    char keypad[4][3] = {{'1','2','3'},
                         {'4','5','6'},
                         {'7','8','9'},
                         {'*','0','#'}};
    printf("Count for numbers of length %d\n", 1, getCount(keypad, 1));
    printf("Count for numbers of length %d\n", 2, getCount(keypad, 2));
    printf("Count for numbers of length %d\n", 3, getCount(keypad, 3));
    printf("Count for numbers of length %d\n", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d\n", 5, getCount(keypad, 5));

    return 0;
}

```

[Run on IDE](#)

Output:

```

Count for numbers of length 1: 10
Count for numbers of length 2: 36
Count for numbers of length 3: 138
Count for numbers of length 4: 532
Count for numbers of length 5: 2062

```

A Space Optimized Solution:

The above dynamic programming approach also runs in $O(n)$ time and requires $O(n)$ auxiliary space, as only one for loop runs n times, other for loops runs for constant time. We can see that n th iteration needs data from $(n-1)$ th iteration only, so we need not keep the data from older iterations. We can have a space efficient dynamic programming approach with just two arrays of size 10. Thanks to Nik for suggesting this solution.

```

// A Space Optimized C program to count number of possible numbers
// of given length
#include <stdio.h>

// Return count of all possible numbers of length n
// in a given numeric keyboard
int getCount(char keypad[][3], int n)
{
    if(keypad == NULL || n <= 0)
        return 0;
    if(n == 1)
        return 10;

    // odd[i], even[i] arrays represent count of numbers starting
    // with digit i for any length j
    int odd[10], even[10];
    int i = 0, j = 0, useOdd = 0, totalCount = 0;

    for (i=0; i<=9; i++)
        odd[i] = 1; // for j = 1

    for (j=2; j<=n; j++) // Bottom Up calculation from j = 2 to n
    {
        useOdd = 1 - useOdd;

        // Here we are explicitly writing lines for each number 0
        // to 9. But it can always be written as DFS on 4X3 grid
        // using row, column array valid moves
        if(useOdd == 1)
        {
            even[0] = odd[0] + odd[8];
            even[1] = odd[1] + odd[2] + odd[4];
            even[2] = odd[2] + odd[1] + odd[3] + odd[5];
        }
        else
        {
            odd[0] = even[0] + even[8];
            odd[1] = even[1] + even[2] + even[4];
            odd[2] = even[2] + even[1] + even[3] + even[5];
        }
    }
    return useOdd ? even[j-1] : odd[j-1];
}

```

```

        even[3] = odd[3] + odd[2] + odd[6];
        even[4] = odd[4] + odd[1] + odd[5] + odd[7];
        even[5] = odd[5] + odd[2] + odd[4] + odd[8] + odd[6];
        even[6] = odd[6] + odd[3] + odd[5] + odd[9];
        even[7] = odd[7] + odd[4] + odd[8];
        even[8] = odd[8] + odd[0] + odd[5] + odd[7] + odd[9];
        even[9] = odd[9] + odd[6] + odd[8];
    }
} else
{
    odd[0] = even[0] + even[8];
    odd[1] = even[1] + even[2] + even[4];
    odd[2] = even[2] + even[1] + even[3] + even[5];
    odd[3] = even[3] + even[2] + even[6];
    odd[4] = even[4] + even[1] + even[5] + even[7];
    odd[5] = even[5] + even[2] + even[4] + even[8] + even[6];
    odd[6] = even[6] + even[3] + even[5] + even[9];
    odd[7] = even[7] + even[4] + even[8];
    odd[8] = even[8] + even[0] + even[5] + even[7] + even[9];
    odd[9] = even[9] + even[6] + even[8];
}
}

// Get count of all possible numbers of length "n" starting
// with digit 0, 1, 2, ..., 9
totalCount = 0;
if(useOdd == 1)
{
    for (i=0; i<=9; i++)
        totalCount += even[i];
}
else
{
    for (i=0; i<=9; i++)
        totalCount += odd[i];
}
return totalCount;
}

// Driver program to test above function
int main()
{
    char keypad[4][3] = {{'1','2','3'},
                         {'4','5','6'},
                         {'7','8','9'},
                         {'*','0','#'}
    };
    printf("Count for numbers of length %d: %d\n", 1, getCount(keypad, 1));
    printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));
    printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));
    printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));

    return 0;
}

```

Run on IDE

Output:

```

Count for numbers of length 1: 10
Count for numbers of length 2: 36
Count for numbers of length 3: 138
Count for numbers of length 4: 532
Count for numbers of length 5: 2062

```

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.



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Matrix](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

4.4

Average Difficulty : 4.4/5.0
Based on 41 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count of n digit numbers whose sum of digits equals to given sum

Given two integers 'n' and 'sum', find count of all n digit numbers with sum of digits as 'sum'. Leading 0's are not counted as digits.

$1 \leq n \leq 100$ and $1 \leq \text{sum} \leq 50000$

Example:

Input: $n = 2$, $\text{sum} = 2$

Output: 2

Explanation: Numbers are 11 and 20

Input: $n = 2$, $\text{sum} = 5$

Output: 5

Explanation: Numbers are 14, 23, 32, 41 and 50

Input: $n = 3$, $\text{sum} = 6$

Output: 21

We strongly recommend that you click here and practice it, before moving on to the solution.

The idea is simple, we subtract all values from 0 to 9 from given sum and recur for sum minus that digit. Below is recursive formula.

```
countRec( $n$ ,  $\text{sum}$ ) =  $\sum \text{finalCount}(n-1, \text{sum}-x)$ 
    where  $1 \leq x \leq 9$  and
           $\text{sum}-x \geq 0$ 
```

One important observation is, leading 0's must be handled explicitly as they are not counted as digits.
So our final count can be written as below.

```
finalCount( $n$ ,  $\text{sum}$ ) =  $\sum \text{finalCount}(n-1, \text{sum}-x)$ 
    where  $0 \leq x \leq 9$  and
           $\text{sum}-x \geq 0$ 
```

Below is a simple recursive solution based on above recursive formula.

```
// A recursive program to count numbers with sum
// of digits as given 'sum'
#include<bits/stdc++.h>
using namespace std;

// Recursive function to count 'n' digit numbers
// with sum of digits as 'sum'. This function
// considers leading 0's also as digits, that is
// why not directly called
unsigned long long int countRec(int n, int sum)
{
    // Base case
    if (n == 0)
        return sum == 0;

    // Initialize answer
    unsigned long long int ans = 0;

    // Traverse through every digit and count
    // numbers beginning with it using recursion
    for (int i=0; i<=9; i++)
        if (sum-i >= 0)
            ans += countRec(n-1, sum-i);

    return ans;
}

// This is mainly a wrapper over countRec. It
// explicitly handles leading digit and calls
// countRec() for remaining digits.
unsigned long long int finalCount(int n, int sum)
{
    // Initialize final answer
    unsigned long long int ans = 0;

    // Traverse through every digit from 1 to
    // 9 and count numbers beginning with it
    for (int i = 1; i <= 9; i++)
        if (sum-i >= 0)
            ans += countRec(n-1, sum-i);

    return ans;
}

// Driver program
int main()
{
    int n = 2, sum = 5;
    cout << finalCount(n, sum);
    return 0;
}
```

Run on IDE

Java

```
class sum_dig
{
    // Recursive function to count 'n' digit numbers
    // with sum of digits as 'sum'. This function
    // considers leading 0's also as digits, that is
    // why not directly called
    static int countRec(int n, int sum)
    {
        // Base case
```

```

if (n == 0)
    return sum == 0 ?1:0;

// Initialize answer
int ans = 0;

// Traverse through every digit and count
// numbers beginning with it using recursion
for (int i=0; i<=9; i++)
    if (sum-i >= 0)
        ans += countRec(n-1, sum-i);

return ans;
}

// This is mainly a wrapper over countRec. It
// explicitly handles leading digit and calls
// countRec() for remaining digits.
static int finalCount(int n, int sum)
{
    // Initialize final answer
    int ans = 0;

    // Traverse through every digit from 1 to
    // 9 and count numbers beginning with it
    for (int i = 1; i <= 9; i++)
        if (sum-i >= 0)
            ans += countRec(n-1, sum-i);

    return ans;
}

/* Driver program to test above function */
public static void main (String args[])
{
    int n = 2, sum = 5;
    System.out.println(finalCount(n, sum));
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

5

The time complexity of above solution is exponential. If we draw the complete recursion tree, we can observe that many subproblems are solved again and again. For example, if we start with $n = 3$ and $sum = 10$, we can reach $n = 1$, $sum = 8$, by considering digit sequences 1,1 or 2, 0.

Since same subproblems are called again, this problem has Overlapping Subproblems property. So min square sum problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem.

Below is Memoization based the implementation.

```

// A memoization based recursive program to count
// numbers with sum of n as given 'sum'
#include<bits/stdc++.h>
using namespace std;

// A lookup table used for memoization
unsigned long long int lookup[101][50001];

// Memoization based implementation of recursive

```

```

// function
unsigned long long int countRec(int n, int sum)
{
    // Base case
    if (n == 0)
        return sum == 0;

    // If this subproblem is already evaluated,
    // return the evaluated value
    if (lookup[n][sum] != -1)
        return lookup[n][sum];

    // Initialize answer
    unsigned long long int ans = 0;

    // Traverse through every digit and
    // recursively count numbers beginning
    // with it
    for (int i=0; i<10; i++)
        if (sum-i >= 0)
            ans += countRec(n-1, sum-i);

    return lookup[n][sum] = ans;
}

// This is mainly a wrapper over countRec. It
// explicitly handles leading digit and calls
// countRec() for remaining n.
unsigned long long int finalCount(int n, int sum)
{
    // Initialize all entries of lookup table
    memset(lookup, -1, sizeof lookup);

    // Initialize final answer
    unsigned long long int ans = 0;

    // Traverse through every digit from 1 to
    // 9 and count numbers beginning with it
    for (int i = 1; i <= 9; i++)
        if (sum-i >= 0)
            ans += countRec(n-1, sum-i);
    return ans;
}

// Driver program
int main()
{
    int n = 3, sum = 5;
    cout << finalCount(n, sum);
    return 0;
}

```

[Run on IDE](#)

Java

```

class sum_dig
{
    // A lookup table used for memoization
    static int lookup[][][] = new int[101][50001];

    // Memoizatiob based implementation of recursive
    // function
    static int countRec(int n, int sum)
    {
        // Base case
        if (n == 0)
            return sum == 0 ? 1 : 0;

        // If this subproblem is already evaluated,
        // return the evaluated value
    }
}

```

```

if (lookup[n][sum] != -1)
    return lookup[n][sum];

// Initialize answer
int ans = 0;

// Traverse through every digit and
// recursively count numbers beginning
// with it
for (int i=0; i<10; i++)
    if (sum-i >= 0)
        ans += countRec(n-1, sum-i);

return lookup[n][sum] = ans;
}

// This is mainly a wrapper over countRec. It
// explicitly handles leading digit and calls
// countRec() for remaining n.
static int finalCount(int n, int sum)
{
    // Initialize all entries of lookup table
    for(int i = 0;i<=100;++i){
        for(int j=0;j<=50000;++j){
            lookup[i][j] = -1;
        }
    }

    // Initialize final answer
    int ans = 0;

    // Traverse through every digit from 1 to
    // 9 and count numbers beginning with it
    for (int i = 1; i <= 9; i++)
        if (sum-i >= 0)
            ans += countRec(n-1, sum-i);
    return ans;
}

/* Driver program to test above function */
public static void main (String args[])
{
    int n = 2, sum = 5;
    System.out.println(finalCount(n, sum));
}
}/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

5

Thanks to [Gaurav Ahirwar](#) for suggesting above solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Mathematical Dynamic Programming number-digits

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.4

Average Difficulty : 3.4/5.0
Based on 36 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Minimum Initial Points to Reach Destination

Given a grid with each cell consisting of positive, negative or no points i.e., zero points. We can move across a cell only if we have positive points (> 0). Whenever we pass through a cell, points in that cell are added to our overall points. We need to find minimum initial points to reach cell $(m-1, n-1)$ from $(0, 0)$.

Constraints :

- From a cell (i, j) we can move to $(i+1, j)$ or $(i, j+1)$.
- We cannot move from (i, j) if your overall points at (i, j) is ≤ 0 .
- We have to reach at $(n-1, m-1)$ with minimum positive points i.e., > 0 .

Example:

```
Input: points[m][n] = { {-2, -3, 3},
                      {-5, -10, 1},
                      {10, 30, -5}
                    };
```

Output: 7

Explanation:

7 is the minimum value to reach destination with positive throughout the path. Below is the path.

$(0,0) \rightarrow (0,1) \rightarrow (0,2) \rightarrow (1, 2) \rightarrow (2, 2)$

We start from $(0, 0)$ with 7, we reach $(0, 1)$ with 5, $(0, 2)$ with 2, $(1, 2)$ with 5, $(2, 2)$ with and finally we have 1 point (we needed greater than 0 points at the end).

We strongly recommend that you click here and practice it, before moving on to the solution.

At the first look, this problem looks similar [Max/Min Cost Path](#), but maximum overall points gained will not guarantee the minimum initial points. Also, it is compulsory in the current problem that the points never drops to zero or below. For instance, Suppose following two paths exists from source to destination cell.

We can solve this problem through bottom-up table filling dynamic programming technique.

- To begin with, we should maintain a 2D array dp of the same size as the grid, where $dp[i][j]$ represents the minimum points that guarantees the continuation of the journey to destination before entering the cell (i, j) . It's but obvious that $dp[0][0]$ is our final solution. Hence, for this problem, we need to fill the table from the bottom right corner to left top.
- Now, let us decide minimum points needed to leave cell (i, j) (remember we are moving from bottom to up). There are only two paths to choose: $(i+1, j)$ and $(i, j+1)$. Of course we will choose the cell that the player can finish the rest of his journey with a smaller initial points. Therefore we have: $\text{min_Points_on_exit} = \min(dp[i+1][j], dp[i][j+1])$

Now we know how to compute min_Points_on_exit, but we need to fill the table $dp[][]$ to get the solution in $dp[0][0]$.

How to compute $dp[i][j]$?

The value of $dp[i][j]$ can be written as below.

$$dp[i][j] = \max(\text{min_Points_on_exit} - \text{points}[i][j], 1)$$

Let us see how above expression covers all cases.

- If $\text{points}[i][j] == 0$, then nothing is gained in this cell; the player can leave the cell with the same points as he enters the room with, i.e. $dp[i][j] = \text{min_Points_on_exit}$.
- If $dp[i][j] < 0$, then the player must have points greater than min_Points_on_exit before entering (i, j) in order to compensate for the points lost in this cell. The minimum amount of compensation is " - points[i][j]", so we have $dp[i][j] = \text{min_Points_on_exit} - \text{points}[i][j]$.
- If $dp[i][j] > 0$, then the player could enter (i, j) with points as little as $\text{min_Points_on_exit} - \text{points}[i][j]$. since he could gain "points[i][j]" points in this cell. However, the value of $\text{min_Points_on_exit} - \text{points}[i][j]$ might drop to 0 or below in this situation. When this happens, we must clip the value to 1 in order to make sure $dp[i][j]$ stays positive:

$$dp[i][j] = \max(\text{min_Points_on_exit} - \text{points}[i][j], 1).$$

Finally return $dp[0][0]$ which is our answer.

Below is the implementation of above algorithm.

```
// C++ program to find minimum initial points to reach destination
#include<bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

int minInitialPoints(int points[][C])
{
    // dp[i][j] represents the minimum initial points player
    // should have so that when starts with cell(i, j) successfully
    // reaches the destination cell(m-1, n-1)
    int dp[R][C];
    int m = R, n = C;

    // Base case
    dp[m-1][n-1] = points[m-1][n-1] > 0? 1:
                    abs(points[m-1][n-1]) + 1;

    // Fill last row and last column as base to fill
    // entire table
    for (int i = m-2; i >= 0; i--)
        dp[i][n-1] = max(dp[i+1][n-1] - points[i][n-1], 1);
    for (int j = n-2; j >= 0; j--)
        dp[m-1][j] = max(dp[m-1][j+1] - points[m-1][j], 1);

    // fill the table in bottom-up fashion
}
```

```

for (int i=m-2; i>=0; i--)
{
    for (int j=n-2; j>=0; j--)
    {
        int min_points_on_exit = min(dp[i+1][j], dp[i][j+1]);
        dp[i][j] = max(min_points_on_exit - points[i][j], 1);
    }
}

return dp[0][0];
}

// Driver Program
int main()
{
    int points[R][C] = { {-2,-3,3},
                        {-5,-10,1},
                        {10,30,-5}
                      };
    cout << "Minimum Initial Points Required: "
          << minInitialPoints(points);
    return 0;
}

```

[Run on IDE](#)

Java

```

class min_steps
{
    static int minInitialPoints(int points[][],int R,int C)
    {
        // dp[i][j] represents the minimum initial points player
        // should have so that when starts with cell(i, j) successfully
        // reaches the destination cell(m-1, n-1)
        int dp[][] = new int[R][C];
        int m = R, n = C;

        // Base case
        dp[m-1][n-1] = points[m-1][n-1] > 0? 1:
                           Math.abs(points[m-1][n-1]) + 1;

        // Fill last row and last column as base to fill
        // entire table
        for (int i = m-2; i >= 0; i--)
            dp[i][n-1] = Math.max(dp[i+1][n-1] - points[i][n-1], 1);
        for (int j = n-2; j >= 0; j--)
            dp[m-1][j] = Math.max(dp[m-1][j+1] - points[m-1][j], 1);

        // fill the table in bottom-up fashion
        for (int i=m-2; i>=0; i--)
        {
            for (int j=n-2; j>=0; j--)
            {
                int min_points_on_exit = Math.min(dp[i+1][j], dp[i][j+1]);
                dp[i][j] = Math.max(min_points_on_exit - points[i][j], 1);
            }
        }

        return dp[0][0];
    }

    /* Driver program to test above function */
    public static void main (String args[])
    {
        int points[][] = { {-2,-3,3},
                          {-5,-10,1},
                          {10,30,-5}
                        };
        int R = 3,C = 3;
    }
}

```

```

        System.out.println("Minimum Initial Points Required: "+  

                           minInitialPoints(points,R,C) );  

    }  
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)**Output:**

```
Minimum Initial Points Required: 7
```

This article is contributed by [Gaurav Ahirwar](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Matrix](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

[\(Login to Rate and Mark\)](#)**4.4**Average Difficulty : **4.4/5.0**
Based on 71 vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Total number of non-decreasing numbers with n digits

A number is non-decreasing if every digit (except the first one) is greater than or equal to previous digit. For example, 223, 4455567, 899, are non-decreasing numbers.

So, given the number of digits n, you are required to find the count of total non-decreasing numbers with n digits.

Examples:

Input: n = 1

Output: count = 10

Input: n = 2

Output: count = 55

Input: n = 3

Output: count = 220

We strongly recommend you to minimize your browser and try this yourself.

One way to look at the problem is, count of numbers is equal to count n digit number ending with 9 plus count of ending with digit 8 plus count for 7 and so on. How to get count ending with a particular digit? We can recur for n-1 length and digits smaller than or equal to the last digit. So below is recursive formula.

```
Count of n digit numbers = (Count of (n-1) digit numbers Ending with digit 9) +
                           (Count of (n-1) digit numbers Ending with digit 8) +
                           .....+
                           .....+
                           (Count of (n-1) digit numbers Ending with digit 0)
```

Let count ending with digit 'd' and length n be count(n, d)

count(n, d) = \sum (count(n-1, i)) where i varies from 0 to d

Total count = \sum count(n-1, d) where d varies from 0 to n-1

The above recursive solution is going to have many overlapping subproblems. Therefore, we can use Dynamic Programming to build a table in bottom up manner. Below is Dynamic programming based C++ program.

```
// C++ program to count non-decreasing number with n digits
```

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

long long int countNonDecreasing(int n)
{
    // dp[i][j] contains total count of non decreasing
    // numbers ending with digit i and of length j
    long long int dp[10][n+1];
    memset(dp, 0, sizeof dp);

    // Fill table for non decreasing numbers of length 1
    // Base cases 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    for (int i = 0; i < 10; i++)
        dp[i][1] = 1;

    // Fill the table in bottom-up manner
    for (int digit = 0; digit <= 9; digit++)
    {
        // Compute total numbers of non decreasing
        // numbers of length 'len'
        for (int len = 2; len <= n; len++)
        {
            // sum of all numbers of length of len-1
            // in which last digit x is <= 'digit'
            for (int x = 0; x <= digit; x++)
                dp[digit][len] += dp[x][len-1];
        }
    }

    long long int count = 0;

    // There total nondecreasing numbers of length n
    // will be dp[0][n] + dp[1][n] ..+ dp[9][n]
    for (int i = 0; i < 10; i++)
        count += dp[i][n];

    return count;
}

// Driver program
int main()
{
    int n = 3;
    cout << countNonDecreasing(n);
    return 0;
}
```

[Run on IDE](#)

Java

```
class NDN
{
    static int countNonDecreasing(int n)
    {
        // dp[i][j] contains total count of non decreasing
        // numbers ending with digit i and of length j
        int dp[][] = new int[10][n+1];

        // Fill table for non decreasing numbers of length 1
        // Base cases 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
        for (int i = 0; i < 10; i++)
            dp[i][1] = 1;

        // Fill the table in bottom-up manner
        for (int digit = 0; digit <= 9; digit++)
        {
            // Compute total numbers of non decreasing
            // numbers of length 'len'
            for (int len = 2; len <= n; len++)
            {

```

```

// sum of all numbers of length of len-1
// in which last digit x is <= 'digit'
for (int x = 0; x <= digit; x++)
    dp[digit][len] += dp[x][len-1];
}

int count = 0;

// There total nondecreasing numbers of length n
// will be dp[0][n] + dp[1][n] ..+ dp[9][n]
for (int i = 0; i < 10; i++)
    count += dp[i][n];

return count;
}
public static void main(String args[])
{
    int n = 3;
    System.out.println(countNonDecreasing(n));
}
/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

220

Thanks to [Gaurav Ahirwar](#) for suggesting above method.

Another method is based on below direct formula

Count of non-decreasing numbers with n digits =

$$N*(N+1)/2*(N+2)/3* \dots * (N+n-1)/n$$

Where N = 10

Below is a C++ program to compute count using above formula.

```

// C++ program to count non-decreasing number with n digits
#include<bits/stdc++.h>
using namespace std;

long long int countNonDecreasing(int n)
{
    int N = 10;

    // Compute value of N*(N+1)/2*(N+2)/3* \dots * (N+n-1)/n
    long long count = 1;
    for (int i=1; i<=n; i++)
    {
        count *= (N+i-1);
        count /= i;
    }

    return count;
}

// Driver program
int main()
{
    int n = 3;
    cout << countNonDecreasing(n);
    return 0;
}

```

[Run on IDE](#)

Output:

220

Thanks to [Abhishek Somani](#) for suggesting this method.

How does this formula work?

$$N * (N+1)/2 * (N+2)/3 * \dots * (N+n-1)/n$$

Where $N = 10$

Let us try for different values of n .

For $n = 1$, the value is N from formula.

Which is true as for $n = 1$, we have all single digit numbers, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

For $n = 2$, the value is $N(N+1)/2$ from formula

We can have N numbers beginning with 0, $(N-1)$ numbers beginning with 1, and so on.

$$\text{So sum is } N + (N-1) + \dots + 1 = N(N+1)/2$$

For $n = 3$, the value is $N(N+1)/2(N+2)/3$ from formula

We can have $N(N+1)/2$ numbers beginning with 0, $(N-1)N/2$ numbers beginning with 1 (Note that when we begin with 1, we have $N-1$ digits left to consider for remaining places), $(N-2)(N-1)/2$ beginning with 2, and so on.

$$\begin{aligned} \text{Count} &= N(N+1)/2 + (N-1)N/2 + (N-2)(N-1)/2 + \\ &\quad (N-3)(N-2)/2 \dots 3 + 1 \\ &[\text{Combining first 2 terms, next 2 terms and so on}] \\ &= 1/2[N^2 + (N-2)^2 + \dots 4] \\ &= N*(N+1)*(N+2)/6 \quad [\text{Refer } \text{this}, \text{ putting } n=N/2 \text{ in the even sum formula}] \end{aligned}$$

For general n digit case, we can apply Mathematical Induction. The count would be equal to count $n-1$ digit beginning with 0, i.e., $N*(N+1)/2*(N+2)/3* \dots *(N+n-1-1)/(n-1)$. Plus count of $n-1$ digit numbers beginning with 1, i.e., $(N-1)*(N)/2*(N+1)/3* \dots *(N-1+n-1-1)/(n-1)$ (Note that N is replaced by $N-1$) and so on.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Mathematical](#) [Dynamic Programming](#) [number-digits](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.4

Average Difficulty : 3.4/5.0
Based on 27 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

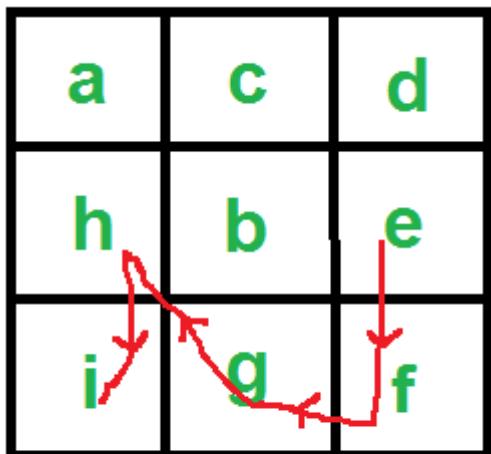
Google™ Custom Search



[Login/Register](#)

Find length of the longest consecutive path from a given starting character

Given a matrix of characters. Find length of the longest path from a given character, such that all characters in the path are consecutive to each other, i.e., every character in path is next to previous in alphabetical order. It is allowed to move in all 8 directions from a cell.



Starting Point 'e'

Example

```
Input: mat[][] = { {a, c, d},
                  {h, b, e},
                  {i, g, f}}
Starting Point = 'e'
```

Output: 5
If starting point is 'e', then longest path with consecutive characters is "e f g h i".

```
Input: mat[R][C] = { {b, e, f},
                  {h, d, a},
                  {i, c, a}};
Starting Point = 'b'
```

Output: 1

'c' is not present in all adjacent cells of 'b'

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to first search given starting character in the given matrix. Do Depth First Search (DFS) from all occurrences to find all consecutive paths. While doing DFS, we may encounter many subproblems again and again. So we use dynamic programming to store results of subproblems.

Below is the implementation of above idea.

```
// C++ program to find the longest consecutive path
#include<bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

// tool matrices to recur for adjacent cells.
int x[] = {0, 1, 1, -1, 1, 0, -1, -1};
int y[] = {1, 0, 1, 1, -1, -1, 0, -1};

// dp[i][j] Stores length of longest consecutive path
// starting at arr[i][j].
int dp[R][C];

// check whether mat[i][j] is a valid cell or not.
bool isvalid(int i, int j)
{
    if (i < 0 || j < 0 || i >= R || j >= C)
        return false;
    return true;
}

// Check whether current character is adjacent to previous
// character (character processed in parent call) or not.
bool isadjacent(char prev, char curr)
{
    return ((curr - prev) == 1);
}

// i, j are the indices of the current cell and prev is the
// character processed in the parent call.. also mat[i][j]
// is our current character.
int getLenUtil(char mat[R][C], int i, int j, char prev)
{
    // If this cell is not valid or current character is not
    // adjacent to previous one (e.g. d is not adjacent to b )
    // or if this cell is already included in the path than return 0.
    if (!isvalid(i, j) || !isadjacent(prev, mat[i][j]))
        return 0;

    // If this subproblem is already solved , return the answer
    if (dp[i][j] != -1)
        return dp[i][j];

    int ans = 0; // Initialize answer

    // recur for paths with differnt adjacent cells and store
    // the length of longest path.
    for (int k=0; k<8; k++)
        ans = max(ans, 1 + getLenUtil(mat, i + x[k],
                                       j + y[k], mat[i][j]));

    // save the answer and return
    return dp[i][j] = ans;
}
```

```

// Returns length of the longest path with all characters consecutive
// to each other. This function first initializes dp array that
// is used to store results of subproblems, then it calls
// recursive DFS based function getLenUtil() to find max length path
int getLen(char mat[R][C], char s)
{
    memset(dp, -1, sizeof dp);
    int ans = 0;

    for (int i=0; i<R; i++)
    {
        for (int j=0; j<C; j++)
        {
            // check for each possible starting point
            if (mat[i][j] == s) {

                // recur for all eight adjacent cells
                for (int k=0; k<8; k++)
                    ans = max(ans, 1 + getLenUtil(mat,
                        i + x[k], j + y[k], s));
            }
        }
    }
    return ans;
}

// Driver program
int main() {

    char mat[R][C] = { { 'a', 'c', 'd' },
                       { 'h', 'b', 'a' },
                       { 'i', 'g', 'f' } };

    cout << getLen(mat, 'a') << endl;
    cout << getLen(mat, 'e') << endl;
    cout << getLen(mat, 'b') << endl;
    cout << getLen(mat, 'f') << endl;
    return 0;
}

```

[Run on IDE](#)

Java

```

class path
{
    // tool matrices to recur for adjacent cells.
    static int x[] = {0, 1, 1, -1, 1, 0, -1, -1};
    static int y[] = {1, 0, 1, 1, -1, -1, 0, -1};
    static int R = 3;
    static int C = 3;
    // dp[i][j] Stores length of longest consecutive path
    // starting at arr[i][j].
    static int dp[][] = new int[R][C];

    // check whether mat[i][j] is a valid cell or not.
    static boolean isValid(int i, int j)
    {
        if (i < 0 || j < 0 || i >= R || j >= C)
            return false;
        return true;
    }

    // Check whether current character is adjacent to previous
    // character (character processed in parent call) or not.
    static boolean isadjacent(char prev, char curr)
    {
        return ((curr - prev) == 1);
    }

    // i, j are the indices of the current cell and prev is the

```

```

// character processed in the parent call.. also mat[i][j]
// is our current character.
static int getLenUtil(char mat[][], int i, int j, char prev)
{
    // If this cell is not valid or current character is not
    // adjacent to previous one (e.g. d is not adjacent to b )
    // or if this cell is already included in the path than return 0.
    if (!isValid(i, j) || !isadjacent(prev, mat[i][j]))
        return 0;

    // If this subproblem is already solved , return the answer
    if (dp[i][j] != -1)
        return dp[i][j];

    int ans = 0; // Initialize answer

    // recur for paths with differnt adjacent cells and store
    // the length of longest path.
    for (int k=0; k<8; k++)
        ans = Math.max(ans, 1 + getLenUtil(mat, i + x[k],
                                            j + y[k], mat[i][j]));

    // save the answer and return
    return dp[i][j] = ans;
}

// Returns length of the longest path with all characters consecutive
// to each other. This function first initializes dp array that
// is used to store results of subproblems, then it calls
// recursive DFS based function getLenUtil() to find max length path
static int getLen(char mat[][], char s)
{
    //assigning all dp values to -1
    for(int i = 0;i<R;++i)
        for(int j = 0;j<C;++j)
            dp[i][j] = -1;

    int ans = 0;

    for (int i=0; i<R; i++)
    {
        for (int j=0; j<C; j++)
        {
            // check for each possible starting point
            if (mat[i][j] == s) {

                // recur for all eight adjacent cells
                for (int k=0; k<8; k++)
                    ans = Math.max(ans, 1 + getLenUtil(mat,
                                                        i + x[k], j + y[k], s));
            }
        }
    }
    return ans;
}

public static void main(String args[])
{
    char mat[][] = { { 'a', 'c', 'd' },
                    { 'h', 'b', 'a' },
                    { 'i', 'g', 'f' } };

    System.out.println(getLen(mat, 'a') );
    System.out.println(getLen(mat, 'e') );
    System.out.println(getLen(mat, 'b') );
    System.out.println(getLen(mat, 'f') );
}
/* This code is contributed by Rajat Mishra */

```

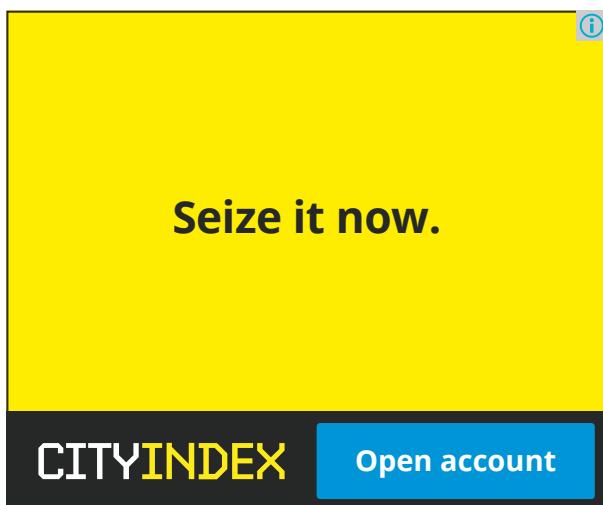
Run on IDE

Output:

```
4
0
3
4
```

Thanks to [Gaurav Ahirwar](#) for above solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Matrix](#) [Dynamic Programming](#) [Matrix](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.3

Average Difficulty : 3.3/5.0
Based on 32 vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Tiling Problem

Given a “ $2 \times n$ ” board and tiles of size “ 2×1 ”, count the number of ways to tile the given board using the 2×1 tiles. A tile can either be placed horizontally i.e., as a 1×2 tile or vertically i.e., as 2×1 tile.

Examples:

Input $n = 3$

Output: 3

Explanation:

We need 3 tiles to tile the board of size 2×3 .

We can tile the board using following ways

- 1) Place all 3 tiles vertically.
- 2) Place first tile vertically and remaining 2 tiles horizontally.
- 3) Place first 2 tiles horizontally and remaining tiles vertically

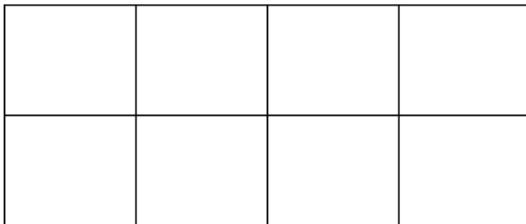
Input $n = 4$

Output: 5

Explanation:

For a 2×4 board, there are 5 ways

- 1) All 4 vertical
- 2) All 4 horizontal
- 3) First 2 vertical, remaining 2 horizontal
- 4) First 2 horizontal, remaining 2 vertical
- 5) Corner 2 vertical, middle 2 horizontal



Board



Tile

We strongly recommend that you click here and practice it, before moving on to the solution.

Let "count(n)" be the count of ways to place tiles on a "2 x n" grid, we have following two ways to place first tile.

- 1) If we place first tile vertically, the problem reduces to "count(n-1)"
- 2) If we place first tile horizontally, we have to place second tile also horizontally. So the problem reduces to "count(n-2)"

Therefore, count(n) can be written as below.

```
count(n) = n if n = 1 or n = 2
count(n) = count(n-1) + count(n-2)
```

The above recurrence is noting but **Fibonacci Number** expression. We can find n'th Fibonacci number in O(Log n) time, see below for all method to find n'th Fibonacci Number.

Different methods for n'th Fibonacci Number.

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



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [MathematicalAlgo](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one

- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.5

Average Difficulty : **2.5/5.0**
Based on **34** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Minimum number of squares whose sum equals to given number n

A number can always be represented as a sum of squares of other numbers. Note that 1 is a square and we can always break a number as $(1^2 + 1^2 + 1^2 + \dots)$. Given a number n, find the minimum number of squares that sum to X.

Examples:

Input: n = 100

Output: 1

100 can be written as 10^2 . Note that 100 can also be written as $5^2 + 5^2 + 5^2 + 5^2$, but this representation requires 4 squares.

Input: n = 6

Output: 3

We strongly recommend you to minimize your browser and try this yourself first.

The idea is simple, we start from 1 and go till a number whose square is smaller than or equals to n. For every number x, we recur for n-x. Below is the recursive formula.

```
If n <= 3, then return n
Else
    minSquares(n) = min {1 + minSquares(n - x*x)}
                        where x >= 1 and x*x <= n
```

Below is a simple recursive solution based on above recursive formula.

```
// A naive recursive C++ program to find minimum
// number of squares whose sum is equal to a given number
#include<bits/stdc++.h>
using namespace std;

// Returns count of minimum squares that sum to n
int getMinSquares(unsigned int n)
{
    // base cases
    if (n <= 3)
        return n;

    // getMinSquares rest of the table using recursive
```

```
// formula
int res = n; // Maximum squares required is n (1*1 + 1*1 + ..)

// Go through all smaller numbers
// to recursively find minimum
for (int x = 1; x <= n; x++)
{
    int temp = x*x;
    if (temp > n)
        break;
    else
        res = min(res, 1+getMinSquares(n - temp));
}
return res;

// Driver program
int main()
{
    cout << getMinSquares(6);
    return 0;
}
```

[Run on IDE](#)

Java

```
// A naive recursive JAVA program to find minimum
// number of squares whose sum is equal to a given number
class squares
{
    // Returns count of minimum squares that sum to n
    static int getMinSquares(int n)
    {
        // base cases
        if (n <= 3)
            return n;

        // getMinSquares rest of the table using recursive
        // formula
        int res = n; // Maximum squares required is n (1*1 + 1*1 + ..)

        // Go through all smaller numbers
        // to recursively find minimum
        for (int x = 1; x <= n; x++)
        {
            int temp = x*x;
            if (temp > n)
                break;
            else
                res = Math.min(res, 1+getMinSquares(n - temp));
        }
        return res;
    }

    public static void main(String args[])
    {
        System.out.println(getMinSquares(6));
    }
}/* This code is contributed by Rajat Mishra */
```

[Run on IDE](#)

Output:

3

The time complexity of above solution is exponential. If we draw the complete recursion tree, we can observe that many subproblems are solved again and again. For example, when we start from $n = 6$, we can reach 4 by subtracting one 2 times and by subtracting 2 one times. So the subproblem for 4 is called twice. Since same subproblems are called again, this problem has Overlapping Subproblems property. So min square sum problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array table[][] in bottom up manner. Below is Dynamic Programming based solution

```
// A dynamic programming based C++ program to find minimum
// number of squares whose sum is equal to a given number
#include<bits/stdc++.h>
using namespace std;

// Returns count of minimum squares that sum to n
int getMinSquares(int n)
{
    // Create a dynamic programming table
    // to store sq
    int *dp = new int[n+1];

    // getMinSquares table for base case entries
    dp[0] = 0;
    dp[1] = 1;
    dp[2] = 2;
    dp[3] = 3;

    // getMinSquares rest of the table using recursive
    // formula
    for (int i = 4; i <= n; i++)
    {
        // max value is i as i can always be represented
        // as 1*1 + 1*1 + ...
        dp[i] = i;

        // Go through all smaller numbers to
        // to recursively find minimum
        for (int x = 1; x <= i; x++) {
            int temp = x*x;
            if (temp > i)
                break;
            else dp[i] = min(dp[i], 1+dp[i-temp]);
        }
    }

    // Store result and free dp[]
    int res = dp[n];
    delete [] dp;

    return res;
}

// Driver program
int main()
{
    cout << getMinSquares(6);
    return 0;
}
```

[Run on IDE](#)

Java

```
// A dynamic programming based JAVA program to find minimum
```

```
// number of squares whose sum is equal to a given number
class squares
{
    // Returns count of minimum squares that sum to n
    static int getMinSquares(int n)
    {
        // Create a dynamic programming table
        // to store sq
        int dp[] = new int[n+1];

        // getMinSquares table for base case entries
        dp[0] = 0;
        dp[1] = 1;
        dp[2] = 2;
        dp[3] = 3;

        // getMinSquares rest of the table using recursive
        // formula
        for (int i = 4; i <= n; i++)
        {
            // max value is i as i can always be represented
            // as 1*1 + 1*1 + ...
            dp[i] = i;

            // Go through all smaller numbers to
            // to recursively find minimum
            for (int x = 1; x <= i; x++) {
                int temp = x*x;
                if (temp > i)
                    break;
                else dp[i] = Math.min(dp[i], 1+dp[i-temp]);
            }
        }

        // Store result and free dp[]
        int res = dp[n];

        return res;
    }
    public static void main(String args[])
    {
        System.out.println(getMinSquares(6));
    }
}/* This code is contributed by Rajat Mishra */
```

[Run on IDE](#)

Output:

3

Thanks to Gaurav Ahirwar for suggesting this solution in a comment [here](#).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Mathematical Dynamic Programming MathematicalAlgo

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 34 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find minimum number of coins that make a given value

Given a value V, if we want to make change for V cents, and we have infinite supply of each of C = {C1, C2, ..., Cm} valued coins, what is the minimum number of coins to make the change?

Examples:

Input: coins[] = {25, 10, 5}, V = 30

Output: Minimum 2 coins required

We can use one coin of 25 cents and one of 5 cents

Input: coins[] = {9, 6, 5, 1}, V = 11

Output: Minimum 2 coins required

We can use one coin of 6 cents and 1 coin of 5 cents

We strongly recommend you to minimize your browser and try this yourself first.

This problem is a variation of the problem discussed [Coin Change Problem](#). Here instead of finding total number of possible solutions, we need to find the solution with minimum number of coins.

The minimum number of coins for a value V can be computed using below recursive formula.

If V == 0, then 0 coins required.

If V > 0

```
minCoin(coins[0..m-1], V) = min {1 + minCoins(V-coin[i])}
                                where i varies from 0 to m-1
                                and coin[i] <= V
```

Below is recursive solution based on above recursive formula.

```
// A Naive recursive C++ program to find minimum of coins
// to make a given change V
#include<bits/stdc++.h>
using namespace std;

// m is size of coins array (number of different coins)
int minCoins(int coins[], int m, int V)
{
    // base case
    if (V == 0) return 0;

    // Initialize result
    int res = INT_MAX;

    // Try every coin as a solution candidate
    for (int i=0; i<m; i++)
        if (coins[i] <= V)
            res = min(res, 1 + minCoins(coins, m, V-coins[i]));
    return res;
}
```

```

int res = INT_MAX;

// Try every coin that has smaller value than V
for (int i=0; i<m; i++)
{
    if (coins[i] <= V)
    {
        int sub_res = minCoins(coins, m, V-coins[i]);

        // Check for INT_MAX to avoid overflow and see if
        // result can minimized
        if (sub_res != INT_MAX && sub_res + 1 < res)
            res = sub_res + 1;
    }
}
return res;
}

// Driver program to test above function
int main()
{
    int coins[] = {9, 6, 5, 1};
    int m = sizeof(coins)/sizeof(coins[0]);
    int V = 11;
    cout << "Minimum coins required is "
         << minCoins(coins, m, V);
    return 0;
}

```

[Run on IDE](#)

Java

```

// A Naive recursive JAVA program to find minimum of coins
// to make a given change V
class coin
{
    // m is size of coins array (number of different coins)
    static int minCoins(int coins[], int m, int V)
    {
        // base case
        if (V == 0) return 0;

        // Initialize result
        int res = Integer.MAX_VALUE;

        // Try every coin that has smaller value than V
        for (int i=0; i<m; i++)
        {
            if (coins[i] <= V)
            {
                int sub_res = minCoins(coins, m, V-coins[i]);

                // Check for INT_MAX to avoid overflow and see if
                // result can minimized
                if (sub_res != Integer.MAX_VALUE && sub_res + 1 < res)
                    res = sub_res + 1;
            }
        }
        return res;
    }

    public static void main(String args[])
    {
        int coins[] = {9, 6, 5, 1};
        int m = coins.length;
        int V = 11;
        System.out.println("Minimum coins required is "+ minCoins(coins, m, V) );
    }
}/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

```
Minimum coins required is 2
```

The time complexity of above solution is exponential. If we draw the complete recursion tree, we can observe that many subproblems are solved again and again. For example, when we start from $V = 11$, we can reach 6 by subtracting one 5 times and by subtracting 5 one times. So the subproblem for 6 is called twice.

Since same subproblems are called again, this problem has Overlapping Subproblems property. So the min coins problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array table[][] in bottom up manner. Below is Dynamic Programming based solution.

```
// A Dynamic Programming based C++ program to find minimum of coins
// to make a given change V
#include<bits/stdc++.h>
using namespace std;

// m is size of coins array (number of different coins)
int minCoins(int coins[], int m, int V)
{
    // table[i] will be storing the minimum number of coins
    // required for i value. So table[V] will have result
    int table[V+1];

    // Base case (If given value V is 0)
    table[0] = 0;

    // Initialize all table values as Infinite
    for (int i=1; i<=V; i++)
        table[i] = INT_MAX;

    // Compute minimum coins required for all
    // values from 1 to V
    for (int i=1; i<=V; i++)
    {
        // Go through all coins smaller than i
        for (int j=0; j<m; j++)
            if (coins[j] <= i)
            {
                int sub_res = table[i-coins[j]];
                if (sub_res != INT_MAX && sub_res + 1 < table[i])
                    table[i] = sub_res + 1;
            }
    }
    return table[V];
}

// Driver program to test above function
int main()
{
    int coins[] = {9, 6, 5, 1};
    int m = sizeof(coins)/sizeof(coins[0]);
    int V = 11;
    cout << "Minimum coins required is "
         << minCoins(coins, m, V);
    return 0;
}
```

Run on IDE

Output:

Minimum coins required is 2

Time complexity of the above solution is $O(mV)$.

Thanks to Goku for suggesting above solution in a comment [here](#) and thanks to Vignesh Mohan for suggesting this problem and initial solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.2

Average Difficulty : 3.2/5.0
Based on 66 vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Collect maximum points in a grid using two traversals

Given a matrix where every cell represents points. How to collect maximum points using two traversals under following conditions?

Let the dimensions of given grid be R x C.

- 1) The first traversal starts from top left corner, i.e., (0, 0) and should reach left bottom corner, i.e., (R-1, 0). The second traversal starts from top right corner, i.e., (0, C-1) and should reach bottom right corner, i.e., (R-1, C-1)/
- 2) From a point (i, j), we can move to (i+1, j+1) or (i+1, j-1) or (i+1, j)
- 3) A traversal gets all points of a particular cell through which it passes. If one traversal has already collected points of a cell, then the other traversal gets no points if goes through that cell again.

Input :

```
int arr[R][C] = {{3, 6, 8, 2},
                  {5, 2, 4, 3},
                  {1, 1, 20, 10},
                  {1, 1, 20, 10},
                  {1, 1, 20, 10},
};
```

Output: 73

Explanation :

3	6	8	2
5	2	4	3
1	1	20	10
1	1	20	10
1	1	20	10

First traversal collects total points of value $3 + 2 + 20 + 1 + 1 = 27$

Second traversal collects total points of value $2 + 4 + 10 + 20 + 10 = 46$.

Total Points collected = $27 + 46 = 73$.

Source: <http://qa.geeksforgeeks.org/1485/running-through-the-grid-to-get-maximum-nutritional-value>

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to do both traversals concurrently. We start first from (0, 0) and second traversal from (0, C-1) simultaneously. The important thing to note is, at any particular step both traversals will be in same row as in all possible three moves, row number is increased. Let (x1, y1) and (x2, y2) denote current positions of first and second traversals respectively. Thus at any time x1 will be equal to x2 as both of them move forward but variation is possible along y. Since variation in y could occur in 3 ways no change (y), go left (y - 1), go right (y + 1). So in total 9 combinations among y1, y2 are possible. The 9 cases as mentioned below after base cases.

```

Both traversals always move forward along x
Base Cases:
// If destinations reached
if (x == R-1 && y1 == 0 && y2 == C-1)
maxPoints(arr, x, y1, y2) = arr[x][y1] + arr[x][y2];

// If any of the two locations is invalid (going out of grid)
if input is not valid
maxPoints(arr, x, y1, y2) = -INF  (minus infinite)

// If both traversals are at same cell, then we count the value of cell
// only once.
If y1 and y2 are same
    result = arr[x][y1]
Else
    result = arr[x][y1] + arr[x][y2]

result += max { // Max of 9 cases
    maxPoints(arr, x+1, y1+1, y2),
    maxPoints(arr, x+1, y1+1, y2+1),
    maxPoints(arr, x+1, y1+1, y2-1),
    maxPoints(arr, x+1, y1-1, y2),
    maxPoints(arr, x+1, y1-1, y2+1),
    maxPoints(arr, x+1, y1-1, y2-1),
    maxPoints(arr, x+1, y1, y2),
    maxPoints(arr, x+1, y1, y2+1),
    maxPoints(arr, x+1, y1, y2-1)
}

```

The above recursive solution has many subproblems that are solved again and again. Therefore, we can use Dynamic Programming to solve the above problem more efficiently. Below is **memoization** (Memoization is alternative to table based iterative solution in Dynamic Programming) based implementation. In below implementation, we use a memoization table 'mem' to keep track of already solved problems.

```

// A Memoization based program to find maximum collection
// using two traversals of a grid
#include<bits/stdc++.h>
using namespace std;
#define R 5
#define C 4

// checks whether a given input is valid or not
bool isValid(int x, int y1, int y2)
{
    return (x >= 0 && x < R && y1 >= 0 &&
            y1 < C && y2 >= 0 && y2 < C);
}

// Driver function to collect max value
int getMaxUtil(int arr[R][C], int mem[R][C][C], int x, int y1, int y2)
{

```

```

/*----- BASE CASES -----*/
// if P1 or P2 is at an invalid cell
if (!isValid(x, y1, y2)) return INT_MIN;

// if both traversals reach their destinations
if (x == R-1 && y1 == 0 && y2 == C-1)
    return arr[x][y1] + arr[x][y2];

// If both traversals are at last row but not at their destination
if (x == R-1) return INT_MIN;

// If subproblem is already solved
if (mem[x][y1][y2] != -1) return mem[x][y1][y2];

// Initialize answer for this subproblem
int ans = INT_MIN;

// this variable is used to store gain of current cell(s)
int temp = (y1 == y2)? arr[x][y1]: arr[x][y1] + arr[x][y2];

/* Recur for all possible cases, then store and return the
   one with max value */
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1, y2-1));
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1, y2+1));
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1, y2));

ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1-1, y2));
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1-1, y2-1));
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1-1, y2+1));

ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1+1, y2));
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1+1, y2-1));
ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1+1, y2+1));

return (mem[x][y1][y2] = ans);
}

// This is mainly a wrapper over recursive function getMaxUtil().
// This function creates a table for memoization and calls
// getMaxUtil()
int geMaxCollection(int arr[R][C])
{
    // Create a memoization table and initialize all entries as -1
    int mem[R][C][C];
    memset(mem, -1, sizeof(mem));

    // Calculation maximum value using memoization based function
    // getMaxUtil()
    return getMaxUtil(arr, mem, 0, 0, C-1);
}

// Driver program to test above functions
int main()
{
    int arr[R][C] = {{3, 6, 8, 2},
                     {5, 2, 4, 3},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10}};
    cout << "Maximum collection is " << geMaxCollection(arr);
    return 0;
}

```

[Run on IDE](#)

Output:

Maximum collection is 73

Thanks to Gaurav Ahirwar for suggesting above problem and solution [here](#).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Matrix Dynamic Programming Matrix

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

4

Average Difficulty : 4/5.0
Based on 56 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Shortest Common Supersequence

Given two strings str1 and str2, find the shortest string that has both str1 and str2 as subsequences.

Examples:

Input: str1 = "geek", str2 = "eke"

Output: "geeke"

Input: str1 = "AGGTAB", str2 = "GXTXAYB"

Output: "AGXGTXAYB"

We strongly recommend you to minimize your browser and try this yourself first.

This problem is closely related to [longest common subsequence problem](#). Below are steps.

1) Find Longest Common Subsequence (lcs) of two given strings. For example, lcs of "geek" and "eke" is "ek".

2) Insert non-lcs characters (in their original order in strings) to the lcs found above, and return the result. So "ek" becomes "geeke" which is shortest common supersequence.

Let us consider another example, str1 = "AGGTAB" and str2 = "GXTXAYB". LCS of str1 and str2 is "GTAB". Once we find LCS, we insert characters of both strings in order and we get "AGXGTXAYB"

How does this work?

We need to find a string that has both strings as subsequences and is shortest such string. If both strings have all characters different, then result is sum of lengths of two given strings. If there are common characters, then we don't want them multiple times as the task is to minimize length. Therefore, we first find the longest common subsequence, take one occurrence of this subsequence and add extra characters.

Length of the shortest supersequence = (Sum of lengths of given two strings) -
 (Length of LCS of two given strings)

Below is C implementation of above idea. The below implementation only finds length of the shortest supersequence.

```
/* C program to find length of the shortest supersequence */
#include<stdio.h>
#include<string.h>

/* Utility function to get max of 2 integers */
int max(int a, int b) { return (a > b)? a : b; }

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n);
```

```

// Function to find length of the shortest supersequence
// of X and Y.
int shortestSuperSequence(char *X, char *Y)
{
    int m = strlen(X), n = strlen(Y);

    int l = lcs(X, Y, m, n); // find lcs

    // Result is sum of input string lengths - length of lcs
    return (m + n - l);
}

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n)
{
    int L[m+1][n+1];
    int i, j;

    /* Following steps build L[m+1][n+1] in bottom up fashion.
       Note that L[i][j] contains length of LCS of X[0..i-1]
       and Y[0..j-1] */
    for (i=0; i<=m; i++)
    {
        for (j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;

            else if (X[i-1] == Y[j-1])
                L[i][j] = L[i-1][j-1] + 1;

            else
                L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }

    /* L[m][n] contains length of LCS for X[0..n-1] and
       Y[0..m-1] */
    return L[m][n];
}

/* Driver program to test above function */
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";
    printf("Length of the shortest supersequence is %d\n",
           shortestSuperSequence(X, Y));
    return 0;
}

```

[Run on IDE](#)

Output:

Length of the shortest supersequence is 9

Below is **Another Method** to solve the above problem.

A simple analysis yields below simple recursive solution.

Let $X[0..m-1]$ and $Y[0..n-1]$ be two strings and m and n be respective lengths.

```
if (m == 0) return n;
```

```

if (n == 0) return m;

// If last characters are same, then add 1 to result and
// recur for X[]
if (X[m-1] == Y[n-1])
    return 1 + SCS(X, Y, m-1, n-1);

// Else find shortest of following two
// a) Remove last character from X and recur
// b) Remove last character from Y and recur
else return 1 + min( SCS(X, Y, m-1, n), SCS(X, Y, m, n-1) );

```

Below is simple naive recursive solution based on above recursive formula.

```

/* A Naive recursive C++ program to find length
   of the shortest supersequence */
#include<bits/stdc++.h>
using namespace std;

int superSeq(char* X, char* Y, int m, int n)
{
    if (!m) return n;
    if (!n) return m;

    if (X[m-1] == Y[n-1])
        return 1 + superSeq(X, Y, m-1, n-1);

    return 1 + min(superSeq(X, Y, m-1, n),
                   superSeq(X, Y, m, n-1));
}

// Driver program to test above function
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";
    cout << "Length of the shortest supersequence is "
         << superSeq(X, Y, strlen(X), strlen(Y));
    return 0;
}

```

[Run on IDE](#)

Output:

Length of the shortest supersequence is 9

Time complexity of the above solution exponential $O(2^{\min(m, n)})$. Since there are **overlapping subproblems**, we can efficiently solve this recursive problem using Dynamic Programming. Below is Dynamic Programming based implementation. Time complexity of this solution is $O(mn)$.

```

/* A dynamic programming based C program to find length
   of the shortest supersequence */
#include<bits/stdc++.h>
using namespace std;

// Returns length of the shortest supersequence of X and Y
int superSeq(char* X, char* Y, int m, int n)
{
    int dp[m+1][n+1];

    // Fill table in bottom up manner
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {

```

```

// Below steps follow above recurrence
if (!i)
    dp[i][j] = j;
else if (!j)
    dp[i][j] = i;
else if (X[i-1] == Y[j-1])
    dp[i][j] = 1 + dp[i-1][j-1];
else
    dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1]);
}

return dp[m][n];
}

// Driver program to test above function
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";
    cout << "Length of the shortest supersequence is "
         << superSeq(X, Y, strlen(X), strlen(Y));
    return 0;
}

```

[Run on IDE](#)**Output:**

Length of the shortest supersequence is 9

Thanks to [Gaurav Ahirwar](#) for suggesting this solution.

Exercise:

Extend the above program to print shortest supersequence also using [function to print LCS](#).

References:

https://en.wikipedia.org/wiki/Shortest_common_supersequence

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Alitile
BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#)[Dynamic Programming](#)[LCS](#)[subsequence](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.2

Average Difficulty : **3.2/5.0**
Based on **25** vote(s)

[Add to TODO List](#)[Mark as DONE](#)

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Compute sum of digits in all numbers from 1 to n

Given a number x, find sum of digits in all numbers from 1 to n.

Examples:

```
Input: n = 5
Output: Sum of digits in numbers from 1 to 5 = 15

Input: n = 12
Output: Sum of digits in numbers from 1 to 12 = 51

Input: n = 328
Output: Sum of digits in numbers from 1 to 328 = 3241
```

Naive Solution:

A naive solution is to go through every number x from 1 to n, and compute sum in x by traversing all digits of x. Below is C++ implementation of this idea.

```
// A Simple C++ program to compute sum of digits in numbers from 1 to n
#include<iostream>
using namespace std;

int sumOfDigits(int );

// Returns sum of all digits in numbers from 1 to n
int sumOfDigitsFrom1ToN(int n)
{
    int result = 0; // initialize result

    // One by one compute sum of digits in every number from
    // 1 to n
    for (int x=1; x<=n; x++)
        result += sumOfDigits(x);

    return result;
}

// A utility function to compute sum of digits in a
// given number x
int sumOfDigits(int x)
{
    int sum = 0;
    while (x != 0)
    {
        sum += x %10;
        x    = x /10;
    }
    return sum;
}

// Driver Program
```

```

int main()
{
    int n = 328;
    cout << "Sum of digits in numbers from 1 to " << n << " is "
        << sumOfDigitsFrom1ToN(n);
    return 0;
}

```

[Run on IDE](#)

Output

Sum of digits in numbers from 1 to 328 is 3241

Efficient Solution:

Above is a naive solution. We can do it more efficiently by finding a pattern.

Let us take few examples.

$$\begin{aligned} \text{sum}(9) &= 1 + 2 + 3 + 4 + \dots + 9 \\ &= 9*10/2 \\ &= 45 \end{aligned}$$

$$\begin{aligned} \text{sum}(99) &= 45 + (10 + 45) + (20 + 45) + \dots + (90 + 45) \\ &= 45*10 + (10 + 20 + 30 + \dots + 90) \\ &= 45*10 + 10(1 + 2 + \dots + 9) \\ &= 45*10 + 45*10 \\ &= \text{sum}(9)*10 + 45*10 \end{aligned}$$

$$\text{sum}(999) = \text{sum}(99)*10 + 45*100$$

In general, we can compute $\text{sum}(10^d - 1)$ using below formula

$$\text{sum}(10^d - 1) = \text{sum}(10^{d-1} - 1) * 10 + 45 * (10^{d-1})$$

In below implementation, the above formula is implemented using **dynamic programming** as there are overlapping subproblems.

The above formula is one core step of the idea. Below is complete algorithm

Algorithm: sum(n)

- 1) Find number of digits minus one in n. Let this value be 'd'.
For 328, d is 2.
- 2) Compute sum of digits in numbers from 1 to $10^d - 1$.
Let this sum be w. For 328, we compute sum of digits from 1 to 99 using above formula.
- 3) Find Most significant digit (msd) in n. For 328, msd is 3.
- 4) Overall sum is sum of following terms
 - a) Sum of digits in 1 to " $\text{msd} * 10^d - 1$ ". For 328, sum of digits in numbers from 1 to 299.
For 328, we compute $3 * \text{sum}(99) + (1 + 2) * 100$. Note that sum of

```
sum(299) is sum(99) + sum of digits from 100 to 199 + sum of digits
from 200 to 299.

Sum of 100 to 199 is sum(99) + 1*100 and sum of 299 is sum(99) + 2*100.

In general, this sum can be computed as w*msd + (msd*(msd-1)/2)*10^d
```

- b) Sum of digits in msd * 10^d to n. For 328, sum of digits in
300 to 328.
For 328, this sum is computed as $3*29 + \text{recursive call "sum(28)"}$
In general, this sum can be computed as $\text{msd} * (\lfloor n / (\text{msd} * 10^d) \rfloor + 1)$
+ $\text{sum}(n \% (\text{msd} * 10^d))$

Below is C++ implementation of above algorithm.

```
// C++ program to compute sum of digits in numbers from 1 to n
#include<bits/stdc++.h>
using namespace std;

// Function to computer sum of digits in numbers from 1 to n
// Comments use example of 328 to explain the code
int sumOfDigitsFrom1ToN(int n)
{
    // base case: if n<10 return sum of
    // first n natural numbers
    if (n<10)
        return n*(n+1)/2;

    // d = number of digits minus one in n. For 328, d is 2
    int d = log10(n);

    // computing sum of digits from 1 to  $10^{d-1}$ ,
    // d=1 a[0]=0;
    // d=2 a[1]=sum of digit from 1 to 9 = 45
    // d=3 a[2]=sum of digit from 1 to 99 = a[1]*10 + 45*10^1 = 900
    // d=4 a[3]=sum of digit from 1 to 999 = a[2]*10 + 45*10^2 = 13500
    int *a = new int[d+1];
    a[0] = 0, a[1] = 45;
    for (int i=2; i<=d; i++)
        a[i] = a[i-1]*10 + 45*ceil(pow(10,i-1));

    // computing  $10^d$ 
    int p = ceil(pow(10, d));

    // Most significant digit (msd) of n,
    // For 328, msd is 3 which can be obtained using 328/100
    int msd = n/p;

    // EXPLANATION FOR FIRST and SECOND TERMS IN BELOW LINE OF CODE
    // First two terms compute sum of digits from 1 to 299
    // (sum of digits in range 1-99 stored in a[d]) +
    // (sum of digits in range 100-199, can be calculated as 1*100 + a[d])
    // (sum of digits in range 200-299, can be calculated as 2*100 + a[d])
    // The above sum can be written as  $3*a[d] + (1+2)*100$ 

    // EXPLANATION FOR THIRD AND FOURTH TERMS IN BELOW LINE OF CODE
    // The last two terms compute sum of digits in number from 300 to 328
    // The third term adds  $3*29$  to sum as digit 3 occurs in all numbers
    // from 300 to 328
    // The fourth term recursively calls for 28
    return msd*a[d] + (msd*(msd-1)/2)*p +
           msd*(1+n%p) + sumOfDigitsFrom1ToN(n%p);
}

// Driver Program
int main()
{
    int n = 328;
    cout << "Sum of digits in numbers from 1 to " << n << " is "
         << sumOfDigitsFrom1ToN(n);
    return 0;
}
```

Output

```
Sum of digits in numbers from 1 to 328 is 3241
```

The efficient algorithm has one more advantage that we need to compute the array ‘a[]’ only once even when we are given multiple inputs.

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

Alitile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Mathematical](#) [Dynamic Programming](#) [MathematicalAlgo](#) [number-digits](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.4

Average Difficulty : 4.4/5.0
Based on 49 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count possible ways to construct buildings

Given an input number of sections and each section has 2 plots on either sides of the road. Find all possible ways to construct buildings in the plots such that there is a space between any 2 buildings.

Example:

```
N = 1
Output = 4
Place a building on one side.
Place a building on other side
Do not place any building.
Place a building on both sides.
```

```
N = 3
Output = 25
3 sections, which means possible ways for one side are
BSS, BSB, SSS, SBS, SSB where B represents a building
and S represents an empty space
Total possible ways are 25, because a way to place on
one side can correspond to any of 5 ways on other side.
```

```
N = 4
Output = 64
```

We strongly recommend to minimize your browser and try this yourself first

We can simplify the problem to first calculate for one side only. If we know the result for one side, we can always do square of the result and get result for two sides.

A new building can be placed on a section if section just before it has space. A space can be placed anywhere (it doesn't matter whether the previous section has a building or not).

```
Let countB(i) be count of possible ways with i sections
ending with a building.
countS(i) be count of possible ways with i sections
ending with a space.
```

```
// A space can be added after a building or after a space.
countS(N) = countB(N-1) + countS(N-1)
```

```
// A building can only be added after a space.
countB[N] = countS(N-1)
```

```
// Result for one side is sum of the above two counts.  
result1(N) = countS(N) + countB(N)  
  
// Result for two sides is square of result1(N)  
result2(N) = result1(N) * result1(N)
```

Below is the implementation of above idea.

```
// C++ program to count all possible way to construct buildings  
#include<iostream>  
using namespace std;  
  
// Returns count of possible ways for N sections  
int countWays(int N)  
{  
    // Base case  
    if (N == 1)  
        return 4; // 2 for one side and 4 for two sides  
  
    // countB is count of ways with a building at the end  
    // countS is count of ways with a space at the end  
    // prev_countB and prev_countS are previous values of  
    // countB and countS respectively.  
  
    // Initialize countB and countS for one side  
    int countB=1, countS=1, prev_countB, prev_countS;  
  
    // Use the above recursive formula for calculating  
    // countB and countS using previous values  
    for (int i=2; i<=N; i++)  
    {  
        prev_countB = countB;  
        prev_countS = countS;  
  
        countS = prev_countB + prev_countS;  
        countB = prev_countS;  
    }  
  
    // Result for one side is sum of ways ending with building  
    // and ending with space  
    int result = countS + countB;  
  
    // Result for 2 sides is square of result for one side  
    return (result*result);  
}  
  
// Driver program  
int main()  
{  
    int N = 3;  
    cout << "Count of ways for " << N  
         << " sections is " << countWays(N);  
    return 0;  
}
```

Java

```
class Building
{
    // Returns count of possible ways for N sections
    static int countWays(int N)
    {
        // Base case
        if (N == 1)
            return 4; // 2 for one side and 4 for two sides

        // countB is count of ways with a building at the end
        // countS is count of ways with a space at the end
        // prev_countB and prev_countS are previous values of
        // countB and countS respectively.

        // Initialize countB and countS for one side
        int countB=1, countS=1, prev_countB, prev_countS;

        // Use the above recursive formula for calculating
        // countB and countS using previous values
        for (int i=2; i<=N; i++)
        {
            prev_countB = countB;
            prev_countS = countS;

            countS = prev_countB + prev_countS;
            countB = prev_countS;
        }

        // Result for one side is sum of ways ending with building
        // and ending with space
        int result = countS + countB;

        // Result for 2 sides is square of result for one side
        return (result*result);
    }

    public static void main(String args[])
    {
        int N = 3;
        System.out.println("Count of ways for "+ N+" sections is "
                           +countWays(N));
    }
}

/* This code is contributed by Rajat Mishra */
```

Output:

25

Time complexity: O(N)

Auxiliary Space: O(1)

Algorithmic Paradigm: Dynamic Programming

Optimized Solution:

Note that the above solution can be further optimized. If we take closer look at the results, for different values, we can notice that the results for two sides are squares of **Fibonacci Numbers**.

N = 1, result = 4 [result for one side = 2]

N = 2, result = 9 [result for one side = 3]

N = 3, result = 25 [result for one side = 5]

N = 4, result = 64 [result for one side = 8]

N = 5, result = 169 [result for one side = 13]

.....

.....

In general, we can say

$$\text{result}(N) = \text{fib}(N+2)^2$$

fib(N) is a function that returns N'th Fibonacci Number.

Therefore, we can use **O(LogN) implementation of Fibonacci Numbers** to find number of ways in **O(logN)** time.

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

GATE CS Corner Company Wise Coding Practice

Dynamic Programming

Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **28** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Maximum profit by buying and selling a share at most twice

In a daily share trading, a buyer buys shares in the morning and sells it on same day. If the trader is allowed to make at most 2 transactions in a day, where as second transaction can only start after first one is complete (Sell->buy->sell->buy). Given stock prices throughout day, find out maximum profit that a share trader could have made.

Examples:

Input: price[] = {10, 22, 5, 75, 65, 80}

Output: 87

Trader earns 87 as sum of 12 and 75

Buy at price 10, sell at 22, buy at 5 and sell at 80

Input: price[] = {2, 30, 15, 10, 8, 25, 80}

Output: 100

Trader earns 100 as sum of 28 and 72

Buy at price 2, sell at 30, buy at 8 and sell at 80

Input: price[] = {100, 30, 15, 10, 8, 25, 80};

Output: 72

Buy at price 8 and sell at 80.

Input: price[] = {90, 80, 70, 60, 50}

Output: 0

Not possible to earn.

We strongly recommend to minimize your browser and try this yourself first.

A **Simple Solution** is to consider every index 'i' and do following

```
Max profit with at most two transactions =
    MAX {max profit with one transaction and subarray price[0..i] +
          max profit with one transaction and subarray price[i+1..n-1] }
i varies from 0 to n-1.
```

Maximum possible using one transaction can be calculated using following O(n) algorithm

Maximum difference between two elements such that larger element appears after the smaller number

Time complexity of above simple solution is O(n²).

We can do this O(n) using following **Efficient Solution**. The idea is to store maximum possible profit of every subarray and solve the problem in following two phases.

1) Create a table profit[0..n-1] and initialize all values in it 0.

2) Traverse price[] from right to left and update profit[i] such that profit[i] stores maximum profit achievable from one transaction in subarray price[i..n-1]

3) Traverse price[] from left to right and update profit[i] such that profit[i] stores maximum profit such that profit[i] contains maximum achievable profit from two transactions in subarray price[0..i].

4) Return profit[n-1]

To do step 1, we need to keep track of maximum price from right to left side and to do step 2, we need to keep track of minimum price from left to right. Why we traverse in reverse directions? The idea is to save space, in second step, we use same array for both purposes, maximum with 1 transaction and maximum with 2 transactions. After an iteration i, the array profit[0..i] contains maximum profit with 2 transactions and profit[i+1..n-1] contains profit with two transactions.

Below are implementations of above idea.

```
// C++ program to find maximum possible profit with at most
// two transactions
#include<iostream>
using namespace std;

// Returns maximum profit with two transactions on a given
// list of stock prices, price[0..n-1]
int maxProfit(int price[], int n)
{
    // Create profit array and initialize it as 0
    int *profit = new int[n];
    for (int i=0; i<n; i++)
        profit[i] = 0;

    /* Get the maximum profit with only one transaction
       allowed. After this loop, profit[i] contains maximum
       profit from price[i..n-1] using at most one trans. */
    int max_price = price[n-1];
    for (int i=n-2;i>=0;i--)
    {
        // max_price has maximum of price[i..n-1]
        if (price[i] > max_price)
            max_price = price[i];

        // we can get profit[i] by taking maximum of:
        // a) previous maximum, i.e., profit[i+1]
        // b) profit by buying at price[i] and selling at
        //     max_price
        profit[i] = max(profit[i+1], max_price-price[i]);
    }

    /* Get the maximum profit with two transactions allowed
       After this loop, profit[n-1] contains the result */
    int min_price = price[0];
    for (int i=1; i<n; i++)
    {
        // min_price is minimum price in price[0..i]
        if (price[i] < min_price)
            min_price = price[i];

        // Maximum profit is maximum of:
        // a) previous maximum, i.e., profit[i-1]
        // b) (Buy, Sell) at (min_price, price[i]) and add
    }
}
```

```

        //    profit of other trans. stored in profit[i]
        profit[i] = max(profit[i-1], profit[i] +
                         (price[i]-min_price) );
    }
    int result = profit[n-1];

    delete [] profit; // To avoid memory leak

    return result;
}

// Drive program
int main()
{
    int price[] = {2, 30, 15, 10, 8, 25, 80};
    int n = sizeof(price)/sizeof(price[0]);
    cout << "Maximum Profit = " << maxProfit(price, n);
    return 0;
}

```

[Run on IDE](#)

Java

```

class Profit
{
    // Returns maximum profit with two transactions on a given
    // list of stock prices, price[0..n-1]
    static int maxProfit(int price[], int n)
    {
        // Create profit array and initialize it as 0
        int profit[] = new int[n];
        for (int i=0; i<n; i++)
            profit[i] = 0;

        /* Get the maximum profit with only one transaction
           allowed. After this loop, profit[i] contains maximum
           profit from price[i..n-1] using at most one trans. */
        int max_price = price[n-1];
        for (int i=n-2;i>=0;i--)
        {
            // max_price has maximum of price[i..n-1]
            if (price[i] > max_price)
                max_price = price[i];

            // we can get profit[i] by taking maximum of:
            // a) previous maximum, i.e., profit[i+1]
            // b) profit by buying at price[i] and selling at
            //     max_price
            profit[i] = Math.max(profit[i+1], max_price-price[i]);
        }

        /* Get the maximum profit with two transactions allowed
           After this loop, profit[n-1] contains the result */
        int min_price = price[0];
        for (int i=1; i<n; i++)
        {
            // min_price is minimum price in price[0..i]
            if (price[i] < min_price)
                min_price = price[i];

            // Maximum profit is maximum of:
            // a) previous maximum, i.e., profit[i-1]
            // b) (Buy, Sell) at (min_price, price[i]) and add
            //     profit of other trans. stored in profit[i]
            profit[i] = Math.max(profit[i-1], profit[i] +
                                (price[i]-min_price) );
        }
        int result = profit[n-1];
        return result;
    }
}

```

```

public static void main(String args[])
{
    int price[] = {2, 30, 15, 10, 8, 25, 80};
    int n = price.length;
    System.out.println("Maximum Profit = " + maxProfit(price, n));
}

/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Python

```

# Returns maximum profit with two transactions on a given
# list of stock prices price[0..n-1]
def maxProfit(price,n):

    # Create profit array and initialize it as 0
    profit = [0]*n

    # Get the maximum profit with only one transaction
    # allowed. After this loop, profit[i] contains maximum
    # profit from price[i..n-1] using at most one trans.
    max_price=price[n-1]

    for i in range( n-2, 0 ,-1):

        if price[i]> max_price:
            max_price = price[i]

        # we can get profit[i] by taking maximum of:
        # a) previous maximum, i.e., profit[i+1]
        # b) profit by buying at price[i] and selling at
        #     max_price
        profit[i] = max(profit[i+1], max_price - price[i])

    # Get the maximum profit with two transactions allowed
    # After this loop, profit[n-1] contains the result
    min_price=price[0]

    for i in range(1,n):

        if price[i] < min_price:
            min_price = price[i]

        # Maximum profit is maximum of:
        # a) previous maximum, i.e., profit[i-1]
        # b) (Buy, Sell) at (min_price, A[i]) and add
        #     profit of other trans. stored in profit[i]
        profit[i] = max(profit[i-1], profit[i]+(price[i]-min_price))

    result = profit[n-1]

    return result

# Driver function
price = [2, 30, 15, 10, 8, 25, 80]
print "Maximum profit is", maxProfit(price, len(price))

# This code is contributed by __Devesh Agrawal__

```

[Run on IDE](#)

Output:

Maximum Profit = 100

Time complexity of the above solution is O(n).

Algorithmic Paradigm: Dynamic Programming

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



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

4.2

Average Difficulty : 4.2/5.0
Based on 70 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

Practice

GATE CS

Placements

GeeksQuiz

Google™ Custom Search



Login/Register

How to print maximum number of A's using given four keys

This is a famous interview question asked in [Google](#), [Paytm](#) and many other company interviews.

Below is the problem statement.

Imagine you have a special keyboard with the following keys:

Key 1: Prints 'A' on screen

Key 2: (Ctrl-A): Select screen

Key 3: (Ctrl-C): Copy selection to buffer

Key 4: (Ctrl-V): Print buffer on screen appending it
after what has already been printed.

If you can only press the keyboard for N times (with the above four keys), write a program to produce maximum numbers of A's. That is to say, the input parameter is N (No. of keys that you can press), the output is M (No. of As that you can produce).

Examples:

Input: N = 3

Output: 3

We can at most get 3 A's on screen by pressing
following key sequence.

A, A, A

Input: N = 7

Output: 9

We can at most get 9 A's on screen by pressing
following key sequence.

A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V

Input: N = 11

Output: 27

We can at most get 27 A's on screen by pressing
following key sequence.

A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V, Ctrl A,
Ctrl C, Ctrl V, Ctrl V

We strongly recommend that you click here and practice it, before moving on to the solution.

Below are few important points to note.

a) For $N < 7$, the output is N itself. b) Ctrl V can be used multiple times to print current buffer (See last two examples above). The idea is to compute the optimal string length for N keystrokes by using a simple insight. The sequence of N keystrokes which produces an optimal string length will end with a suffix of Ctrl-A, a Ctrl-C, followed by only Ctrl-V's (For $N > 6$).

The task is to find out the break-point after which we get the above suffix of keystrokes. Definition of a breakpoint is that instance after which we need to only press Ctrl-A, Ctrl-C once and the only Ctrl-V's afterwards to generate the optimal length. If we loop from $N-3$ to 1 and choose each of these values for the break-point, and compute that optimal string they would produce. Once the loop ends, we will have the maximum of the optimal lengths for various breakpoints, thereby giving us the optimal length for N keystrokes.

Below is C implementation based on above idea.

```
/* A recursive C program to print maximum number of A's using
   following four keys */
#include<stdio.h>

// A recursive function that returns the optimal length string
// for N keystrokes
int findoptimal(int N)
{
    // The optimal string length is N when N is smaller than 7
    if (N <= 6)
        return N;

    // Initialize result
    int max = 0;

    // TRY ALL POSSIBLE BREAK-POINTS
    // For any keystroke N, we need to loop from N-3 keystrokes
    // back to 1 keystroke to find a breakpoint 'b' after which we
    // will have Ctrl-A, Ctrl-C and then only Ctrl-V all the way.
    int b;
    for (b=N-3; b>=1; b--)
    {
        // If the breakpoint is s at b'th keystroke then
        // the optimal string would have length
        // (n-b-1)*screen[b-1];
        int curr = (N-b-1)*findoptimal(b);
        if (curr > max)
            max = curr;
    }
    return max;
}

// Driver program
int main()
{
    int N;

    // for the rest of the array we will rely on the previous
    // entries to compute new ones
    for (N=1; N<=20; N++)
        printf("Maximum Number of A's with %d keystrokes is %d\n",
               N, findoptimal(N));
}
```

Run on IDE

Output:

```

Maximum Number of A's with 1 keystrokes is 1
Maximum Number of A's with 2 keystrokes is 2
Maximum Number of A's with 3 keystrokes is 3
Maximum Number of A's with 4 keystrokes is 4
Maximum Number of A's with 5 keystrokes is 5
Maximum Number of A's with 6 keystrokes is 6
Maximum Number of A's with 7 keystrokes is 9
Maximum Number of A's with 8 keystrokes is 12
Maximum Number of A's with 9 keystrokes is 16
Maximum Number of A's with 10 keystrokes is 20
Maximum Number of A's with 11 keystrokes is 27
Maximum Number of A's with 12 keystrokes is 36
Maximum Number of A's with 13 keystrokes is 48
Maximum Number of A's with 14 keystrokes is 64
Maximum Number of A's with 15 keystrokes is 81
Maximum Number of A's with 16 keystrokes is 108
Maximum Number of A's with 17 keystrokes is 144
Maximum Number of A's with 18 keystrokes is 192
Maximum Number of A's with 19 keystrokes is 256
Maximum Number of A's with 20 keystrokes is 324

```

The above function computes the same subproblems again and again. Recomputations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

Below is Dynamic Programming based C implementation where an auxiliary array screen[N] is used to store result of subproblems.

```

/* A Dynamic Programming based C program to find maximum number of A's
   that can be printed using four keys */
#include<stdio.h>

// this function returns the optimal length string for N keystrokes
int findoptimal(int N)
{
    // The optimal string length is N when N is smaller than 7
    if (N <= 6)
        return N;

    // An array to store result of subproblems
    int screen[N];

    int b; // To pick a breakpoint

    // Initializing the optimal lengths array for upto 6 input
    // strokes.
    int n;
    for (n=1; n<=6; n++)
        screen[n-1] = n;

    // Solve all subproblems in bottom manner
    for (n=7; n<=N; n++)
    {
        // Initialize length of optimal string for n keystrokes
        screen[n-1] = 0;

        // For any keystroke n, we need to loop from n-3 keystrokes
        // back to 1 keystroke to find a breakpoint 'b' after which we
        // will have ctrl-a, ctrl-c and then only ctrl-v all the way.
        for (b=n-3; b>=1; b--)
        {
            // if the breakpoint is at b'th keystroke then
            // the optimal string would have length
            // (n-b-1)*screen[b-1];
            int curr = (n-b-1)*screen[b-1];
            if (curr > screen[n-1])

```

```
        screen[n-1] = curr;
    }

    return screen[N-1];
}

// Driver program
int main()
{
    int N;

    // for the rest of the array we will rely on the previous
    // entries to compute new ones
    for (N=1; N<=20; N++)
        printf("Maximum Number of A's with %d keystrokes is %d\n",
               N, findoptimal(N));
}
```

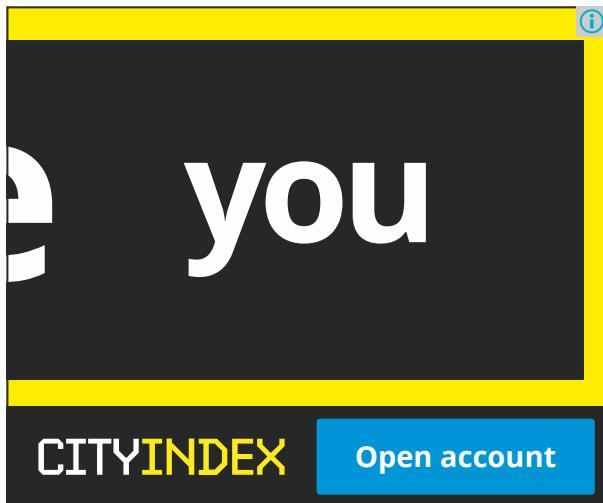
[Run on IDE](#)

Output:

```
Maximum Number of A's with 1 keystrokes is 1
Maximum Number of A's with 2 keystrokes is 2
Maximum Number of A's with 3 keystrokes is 3
Maximum Number of A's with 4 keystrokes is 4
Maximum Number of A's with 5 keystrokes is 5
Maximum Number of A's with 6 keystrokes is 6
Maximum Number of A's with 7 keystrokes is 9
Maximum Number of A's with 8 keystrokes is 12
Maximum Number of A's with 9 keystrokes is 16
Maximum Number of A's with 10 keystrokes is 20
Maximum Number of A's with 11 keystrokes is 27
Maximum Number of A's with 12 keystrokes is 36
Maximum Number of A's with 13 keystrokes is 48
Maximum Number of A's with 14 keystrokes is 64
Maximum Number of A's with 15 keystrokes is 81
Maximum Number of A's with 16 keystrokes is 108
Maximum Number of A's with 17 keystrokes is 144
Maximum Number of A's with 18 keystrokes is 192
Maximum Number of A's with 19 keystrokes is 256
Maximum Number of A's with 20 keystrokes is 324
```

Thanks to **Gaurav Saxena** for providing the above approach to solve this problem.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

4.4

Average Difficulty : **4.4/5.0**
Based on **90** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Find the minimum cost to reach destination using a train

There are N stations on route of a train. The train goes from station 0 to N-1. The ticket cost for all pair of stations (i, j) is given where j is greater than i. Find the minimum cost to reach the destination.

Consider the following example:

Input:

```
cost[N][N] = { {0, 15, 80, 90},
               {INF, 0, 40, 50},
               {INF, INF, 0, 70},
               {INF, INF, INF, 0}
             };
```

There are 4 stations and cost[i][j] indicates cost to reach j from i. The entries where j < i are meaningless.

Output:

The minimum cost is 65

The minimum cost can be obtained by first going to station 1 from 0. Then from station 1 to station 3.

We strongly recommend to minimize your browser and try this yourself first.

The minimum cost to reach N-1 from 0 can be recursively written as following:

```
minCost(0, N-1) = MIN { cost[0][n-1],
                        cost[0][1] + minCost(1, N-1),
                        minCost(0, 2) + minCost(2, N-1),
                        ....,
                        minCost(0, N-2) + cost[N-2][n-1] }
```

The following is the implementation of above recursive formula.

```
// A naive recursive solution to find min cost path from station 0
// to station N-1
#include<iostream>
#include<climits>
using namespace std;

// infinite value
#define INF INT_MAX
```

```

// Number of stations
#define N 4

// A recursive function to find the shortest path from
// source 's' to destination 'd'.
int minCostRec(int cost[][N], int s, int d)
{
    // If source is same as destination
    // or destination is next to source
    if (s == d || s+1 == d)
        return cost[s][d];

    // Initialize min cost as direct ticket from
    // source 's' to destination 'd'.
    int min = cost[s][d];

    // Try every intermediate vertex to find minimum
    for (int i = s+1; i<d; i++)
    {
        int c = minCostRec(cost, s, i) +
                minCostRec(cost, i, d);
        if (c < min)
            min = c;
    }
    return min;
}

// This function returns the smallest possible cost to
// reach station N-1 from station 0. This function mainly
// uses minCostRec().
int minCost(int cost[][])
{
    return minCostRec(cost, 0, N-1);
}

// Driver program to test above function
int main()
{
    int cost[N][N] = { {0, 15, 80, 90},
                      {INF, 0, 40, 50},
                      {INF, INF, 0, 70},
                      {INF, INF, INF, 0}
                    };
    cout << "The Minimum cost to reach station "
         << N << " is " << minCost(cost);
    return 0;
}

```

[Run on IDE](#)

Java

```

// A naive recursive solution to find min cost path from station 0
// to station N-1
class shortest_path
{

    static int INF = Integer.MAX_VALUE,N = 4;
    // A recursive function to find the shortest path from
    // source 's' to destination 'd'.
    static int minCostRec(int cost[][], int s, int d)
    {
        // If source is same as destination
        // or destination is next to source
        if (s == d || s+1 == d)
            return cost[s][d];

        // Initialize min cost as direct ticket from
        // source 's' to destination 'd'.
        int min = cost[s][d];

```

```

// Try every intermediate vertex to find minimum
for (int i = s+1; i<d; i++)
{
    int c = minCostRec(cost, s, i) +
            minCostRec(cost, i, d);
    if (c < min)
        min = c;
}
return min;
}

// This function returns the smallest possible cost to
// reach station N-1 from station 0. This function mainly
// uses minCostRec().
static int minCost(int cost[][][])
{
    return minCostRec(cost, 0, N-1);
}

public static void main(String args[])
{
    int cost[][] = { {0, 15, 80, 90},
                    {INF, 0, 40, 50},
                    {INF, INF, 0, 70},
                    {INF, INF, INF, 0}
                };
    System.out.println("The Minimum cost to reach station "+ N+
                        " is "+minCost(cost));
}

/* This code is contributed by Rajat Mishra */

```

[Run on IDE](#)

Output:

The Minimum cost to reach station 4 is 65

Time complexity of the above implementation is exponential as it tries every possible path from 0 to N-1. The above solution solves same subproblems multiple times (it can be seen by drawing recursion tree for minCostPathRec(0, 5)).

Since this problem has both properties of dynamic programming problems ((see [this](#) and [this](#)). Like other typical Dynamic Programming(DP) problems, re-computations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

One dynamic programming solution is to create a 2D table and fill the table using above given recursive formula. The extra space required in this solution would be $O(N^2)$ and time complexity would be $O(N^3)$

We can solve this problem using $O(N)$ extra space and $O(N^2)$ time. The idea is based on the fact that given input matrix is a Directed Acyclic Graph (DAG). The shortest path in DAG can be calculated using the approach discussed in below post.

Shortest Path in Directed Acyclic Graph

We need to do less work here compared to above mentioned post as we know [topological sorting](#) of the graph. The topological sorting of vertices here is 0, 1, ..., N-1. Following is the idea once topological sorting is known.

The idea in below code is to first calculate min cost for station 1, then for station 2, and so on. These costs are stored in an array dist[0...N-1].

- 1) The min cost for station 0 is 0, i.e., dist[0] = 0

2) The min cost for station 1 is $\text{cost}[0][1]$, i.e., $\text{dist}[1] = \text{cost}[0][1]$

3) The min cost for station 2 is minimum of following two.

- a) $\text{dist}[0] + \text{cost}[0][2]$
- b) $\text{dist}[1] + \text{cost}[1][2]$

3) The min cost for station 3 is minimum of following three.

- a) $\text{dist}[0] + \text{cost}[0][3]$
- b) $\text{dist}[1] + \text{cost}[1][3]$
- c) $\text{dist}[2] + \text{cost}[2][3]$

Similarly, $\text{dist}[4]$, $\text{dist}[5]$, ... $\text{dist}[N-1]$ are calculated.

Below is the implementation of above idea.

```
// A Dynamic Programming based solution to find min cost
// to reach station N-1 from station 0.
#include<iostream>
#include<climits>
using namespace std;

#define INF INT_MAX
#define N 4

// This function returns the smallest possible cost to
// reach station N-1 from station 0.
int minCost(int cost[][])
{
    // dist[i] stores minimum cost to reach station i
    // from station 0.
    int dist[N];
    for (int i=0; i<N; i++)
        dist[i] = INF;
    dist[0] = 0;

    // Go through every station and check if using it
    // as an intermediate station gives better path
    for (int i=0; i<N; i++)
        for (int j=i+1; j<N; j++)
            if (dist[j] > dist[i] + cost[i][j])
                dist[j] = dist[i] + cost[i][j];

    return dist[N-1];
}

// Driver program to test above function
int main()
{
    int cost[N][N] = { {0, 15, 80, 90},
                      {INF, 0, 40, 50},
                      {INF, INF, 0, 70},
                      {INF, INF, INF, 0}
                    };
    cout << "The Minimum cost to reach station "
         << N << " is " << minCost(cost);
    return 0;
}
```

Run on IDE

Java

```
// A Dynamic Programming based solution to find min cost
```

```
// to reach station N-1 from station 0.
class shortest_path
{

    static int INF = Integer.MAX_VALUE, N = 4;
    // A recursive function to find the shortest path from
    // source 's' to destination 'd'.

    // This function returns the smallest possible cost to
    // reach station N-1 from station 0.
    static int minCost(int cost[][])
    {
        // dist[i] stores minimum cost to reach station i
        // from station 0.
        int dist[] = new int[N];
        for (int i=0; i<N; i++)
            dist[i] = INF;
        dist[0] = 0;

        // Go through every station and check if using it
        // as an intermediate station gives better path
        for (int i=0; i<N; i++)
            for (int j=i+1; j<N; j++)
                if (dist[j] > dist[i] + cost[i][j])
                    dist[j] = dist[i] + cost[i][j];

        return dist[N-1];
    }

    public static void main(String args[])
    {
        int cost[][] = { {0, 15, 80, 90},
                        {INF, 0, 40, 50},
                        {INF, INF, 0, 70},
                        {INF, INF, INF, 0}
                      };
        System.out.println("The Minimum cost to reach station "+ N+
                           " is "+minCost(cost));
    }
}/* This code is contributed by Rajat Mishra */
```

[Run on IDE](#)

Output:

The Minimum cost to reach station 4 is 65

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

Ali Tile BY **LOLA GROUP**

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Graph](#) [shortest path](#) [Topological Sorting](#)

Related Posts:

- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence

(Login to Rate and Mark)

2.8

Average Difficulty : **2.8/5.0**
Based on **38** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



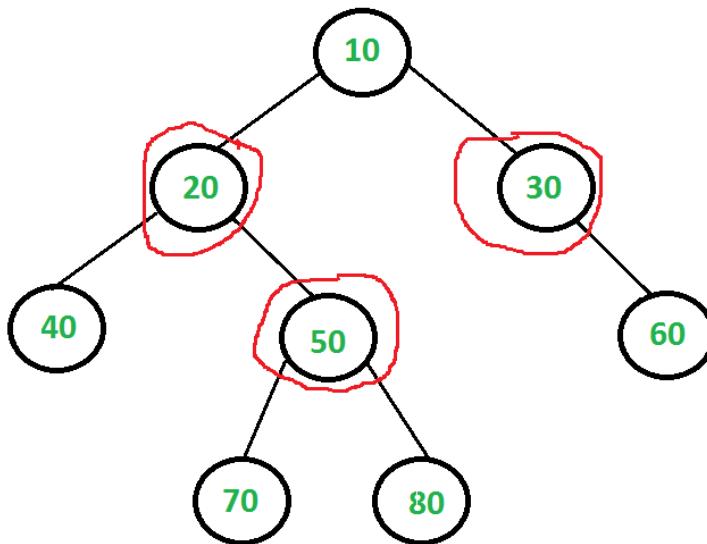
[Login/Register](#)

Vertex Cover Problem | Set 2 (Dynamic Programming Solution for Tree)

A [vertex cover](#) of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either ' u ' or ' v ' is in vertex cover. Although the name is Vertex Cover, the set covers all edges of the given graph.

The problem to find minimum size vertex cover of a graph is [NP complete](#). But it can be solved in polynomial time for trees. In this post a solution for Binary Tree is discussed. The same solution can be extended for n-ary trees.

For example, consider the following binary tree. The smallest vertex cover is $\{20, 50, 30\}$ and size of the vertex cover is 3.



The idea is to consider following two possibilities for root and recursively for all nodes down the root.

1) Root is part of vertex cover: In this case root covers all children edges. We recursively calculate size of vertex covers for left and right subtrees and add 1 to the result (for root).

2) Root is not part of vertex cover: In this case, both children of root must be included in vertex cover to cover all root to children edges. We recursively calculate size of vertex covers of all grandchildren and number of children to the result (for two children of root).

Below is C implementation of above idea.

```

// A naive recursive C implementation for vertex cover problem for a tree
#include <stdio.h>
#include <stdlib.h>

// A utility function to find min of two integers
int min(int x, int y) { return (x < y)? x: y; }
  
```

```

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    struct node *left, *right;
};

// The function returns size of the minimum vertex cover
int vCover(struct node *root)
{
    // The size of minimum vertex cover is zero if tree is empty or there
    // is only one node
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return 0;

    // Calculate size of vertex cover when root is part of it
    int size_incl = 1 + vCover(root->left) + vCover(root->right);

    // Calculate size of vertex cover when root is not part of it
    int size_excl = 0;
    if (root->left)
        size_excl += 1 + vCover(root->left->left) + vCover(root->left->right);
    if (root->right)
        size_excl += 1 + vCover(root->right->left) + vCover(root->right->right);

    // Return the minimum of two sizes
    return min(size_incl, size_excl);
}

// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root      = newNode(20);
    root->left           = newNode(8);
    root->left->left     = newNode(4);
    root->left->right    = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    root->right          = newNode(22);
    root->right->right   = newNode(25);

    printf ("Size of the smallest vertex cover is %d ", vCover(root));

    return 0;
}

```

[Run on IDE](#)

Output:

Size of the smallest vertex cover is 3

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. For example, vCover of node with value 50 is evaluated twice as 50 is grandchild of 10 and child of 20.

Since same subproblems are called again, this problem has Overlapping Subproblems property. So Vertex Cover

problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), re-computations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

Following is C implementation of Dynamic Programming based solution. In the following solution, an additional field 'vc' is added to tree nodes. The initial value of 'vc' is set as 0 for all nodes. The recursive function vCover() calculates 'vc' for a node only if it is not already set.

```
/* Dynamic programming based program for Vertex Cover problem for
   a Binary Tree */
#include <stdio.h>
#include <stdlib.h>

// A utility function to find min of two integers
int min(int x, int y) { return (x < y)? x: y; }

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    int vc;
    struct node *left, *right;
};

// A memoization based function that returns size of the minimum vertex cover.
int vCover(struct node *root)
{
    // The size of minimum vertex cover is zero if tree is empty or there
    // is only one node
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return 0;

    // If vertex cover for this node is already evaluated, then return it
    // to save recomputation of same subproblem again.
    if (root->vc != 0)
        return root->vc;

    // Calculate size of vertex cover when root is part of it
    int size_incl = 1 + vCover(root->left) + vCover(root->right);

    // Calculate size of vertex cover when root is not part of it
    int size_excl = 0;
    if (root->left)
        size_excl += 1 + vCover(root->left->left) + vCover(root->left->right);
    if (root->right)
        size_excl += 1 + vCover(root->right->left) + vCover(root->right->right);

    // Minimum of two values is vertex cover, store it before returning
    root->vc = min(size_incl, size_excl);

    return root->vc;
}

// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    temp->vc = 0; // Set the vertex cover as 0
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root          = newNode(20);
```

```

root->left          = newNode(8);
root->left->left   = newNode(4);
root->left->right  = newNode(12);
root->left->right->left = newNode(10);
root->left->right->right = newNode(14);
root->right         = newNode(22);
root->right->right = newNode(25);

printf ("Size of the smallest vertex cover is %d ", vCover(root));

return 0;
}

```

[Run on IDE](#)

Output:

Size of the smallest vertex cover is 3

References:

<http://courses.csail.mit.edu/6.006/spring11/lectures/lec21.pdf>

Exercise:

Extend the above solution for n-ary trees.

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

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#) [NP Hard](#)

Related Posts:

- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets

- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 21 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Count number of ways to reach a given score in a game

Consider a game where a player can score 3 or 5 or 10 points in a move. Given a total score n, find number of ways to reach the given score.

Examples:

Input: n = 20

Output: 4

There are following 4 ways to reach 20

(10, 10)

(5, 5, 10)

(5, 5, 5, 5)

(3, 3, 3, 3, 3, 5)

Input: n = 13

Output: 2

There are following 2 ways to reach 13

(3, 5, 5)

(3, 10)

We strongly recommend that you click here and practice it, before moving on to the solution.

This problem is a variation of [coin change problem](#) and can be solved in O(n) time and O(n) auxiliary space.

The idea is to create a table of size n+1 to store counts of all scores from 0 to n. For every possible move (3, 5 and 10), increment values in table.

```
// A C program to count number of possible ways to a given score
// can be reached in a game where a move can earn 3 or 5 or 10
#include <stdio.h>

// Returns number of ways to reach score n
int count(int n)
{
    // table[i] will store count of solutions for
    // value i.
    int table[n+1], i;

    // Initialize all table values as 0
    memset(table, 0, sizeof(table));

    // Base case (If given value is 0)
    if (n == 0)
        return 1;
    if (n < 0)
        return 0;

    // Count ways by considering
    // all three moves 3, 5 and 10
    for (i = 1; i <= n; i++)
    {
        if (i - 3 >= 0)
            table[i] += table[i - 3];
        if (i - 5 >= 0)
            table[i] += table[i - 5];
        if (i - 10 >= 0)
            table[i] += table[i - 10];
    }
    return table[n];
}
```

```
table[0] = 1;

// One by one consider given 3 moves and update the table[]
// values after the index greater than or equal to the
// value of the picked move
for (i=3; i<=n; i++)
    table[i] += table[i-3];
for (i=5; i<=n; i++)
    table[i] += table[i-5];
for (i=10; i<=n; i++)
    table[i] += table[i-10];

return table[n];
}

// Driver program
int main(void)
{
    int n = 20;
    printf("Count for %d is %d\n", n, count(n));

    n = 13;
    printf("Count for %d is %d", n, count(n));
    return 0;
}
```

Run on IDE

Output:

Count for 20 is 4
Count for 13 is 2

Exercise: How to count score when $(10, 5, 5)$, $(5, 5, 10)$ and $(5, 10, 5)$ are considered as different sequences of moves. Similarly, $(5, 3, 3)$, $(3, 5, 3)$ and $(3, 3, 5)$ are considered different.

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



GATE CS Corner Company Wise Coding Practice

Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

2.5

Average Difficulty : 2.5/5.0
Based on 33 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Weighted Job Scheduling

Given N jobs where every job is represented by following three elements of it.

1. Start Time
2. Finish Time
3. Profit or Value Associated

Find the maximum profit subset of jobs such that no two jobs in the subset overlap.

Example:

```
Input: Number of Jobs n = 4
Job Details {Start Time, Finish Time, Profit}
Job 1: {1, 2, 50}
Job 2: {3, 5, 20}
Job 3: {6, 19, 100}
Job 4: {2, 100, 200}
```

Output: The maximum profit is 250.

We can get the maximum profit by scheduling jobs 1 and 4.

Note that there is longer schedules possible Jobs 1, 2 and 3

but the profit with this schedule is 20+50+100 which is less than 250.

A simple version of this problem is discussed [here](#) where every job has same profit or value. The [Greedy Strategy for activity selection](#) doesn't work here as the longer schedule may have smaller profit or value.

The above problem can be solved using following recursive solution.

```
1) First sort jobs according to finish time.
2) Now apply following recursive process.
// Here arr[] is array of n jobs
findMaximumProfit(arr[], n)
{
    a) if (n == 1) return arr[0];
    b) Return the maximum of following two profits.
        (i) Maximum profit by excluding current job, i.e.,
            findMaximumProfit(arr, n-1)
        (ii) Maximum profit by including the current job
}
```

How to find the profit including current job?

The idea is to find the latest job before the current job (in sorted array) that doesn't conflict with current job 'arr[n-1]'.

Once we find such a job, we recur for all jobs till that job and add profit of current job to result.
 In the above example, "job 1" is the latest non-conflicting for "job 4" and "job 2" is the latest non-conflicting for "job 3".

The following is C++ implementation of above naive recursive method.

```
// C++ program for weighted job scheduling using Naive Recursive Method
#include <iostream>
#include <algorithm>
using namespace std;

// A job has start time, finish time and profit.
struct Job
{
    int start, finish, profit;
};

// A utility function that is used for sorting events
// according to finish time
bool jobComparataor(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}

// Find the latest job (in sorted array) that doesn't
// conflict with the job[i]. If there is no compatible job,
// then it returns -1.
int latestNonConflict(Job arr[], int i)
{
    for (int j=i-1; j>=0; j--)
    {
        if (arr[j].finish <= arr[i-1].start)
            return j;
    }
    return -1;
}

// A recursive function that returns the maximum possible
// profit from given array of jobs. The array of jobs must
// be sorted according to finish time.
int findMaxProfitRec(Job arr[], int n)
{
    // Base case
    if (n == 1) return arr[n-1].profit;

    // Find profit when current job is included
    int inclProf = arr[n-1].profit;
    int i = latestNonConflict(arr, n);
    if (i != -1)
        inclProf += findMaxProfitRec(arr, i+1);

    // Find profit when current job is excluded
    int exclProf = findMaxProfitRec(arr, n-1);

    return max(inclProf, exclProf);
}

// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit(Job arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, jobComparataor);

    return findMaxProfitRec(arr, n);
}

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

```

Job arr[] = {{3, 10, 20}, {1, 2, 50}, {6, 19, 100}, {2, 100, 200}};
int n = sizeof(arr)/sizeof(arr[0]);
cout << "The optimal profit is " << findMaxProfit(arr, n);
return 0;
}

```

[Run on IDE](#)

Output:

```
The optimal profit is 250
```

The above solution may contain many overlapping subproblems. For example if `lastNonConflicting()` always returns previous job, then `findMaxProfitRec(arr, n-1)` is called twice and the time complexity becomes $O(n^2^n)$. As another example when `lastNonConflicting()` returns previous to previous job, there are two recursive calls, for $n-2$ and $n-1$. In this example case, recursion becomes same as Fibonacci Numbers.

So this problem has both properties of Dynamic Programming, [Optimal Substructure](#) and [Overlapping Subproblems](#).

Like other Dynamic Programming Problems, we can solve this problem by making a table that stores solution of subproblems.

Below is C++ implementation based on Dynamic Programming.

```

// C++ program for weighted job scheduling using Dynamic Programming.
#include <iostream>
#include <algorithm>
using namespace std;

// A job has start time, finish time and profit.
struct Job
{
    int start, finish, profit;
};

// A utility function that is used for sorting events
// according to finish time
bool jobComparataor(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}

// Find the latest job (in sorted array) that doesn't
// conflict with the job[i]
int latestNonConflict(Job arr[], int i)
{
    for (int j=i-1; j>=0; j--)
    {
        if (arr[j].finish <= arr[i].start)
            return j;
    }
    return -1;
}

// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit(Job arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, jobComparataor);

    // Create an array to store solutions of subproblems. table[i]
    // stores the profit for jobs till arr[i] (including arr[i])
    int *table = new int[n];
    table[0] = arr[0].profit;

    // Fill entries in M[] using recursive property
    for (int i=1; i<n; i++)

```

```

    // Find profit including the current job
    int inclProf = arr[i].profit;
    int l = latestNonConflict(arr, i);
    if (l != -1)
        inclProf += table[l];

    // Store maximum of including and excluding
    table[i] = max(inclProf, table[i-1]);
}

// Store result and free dynamic memory allocated for table[]
int result = table[n-1];
delete[] table;

return result;
}

// Driver program
int main()
{
    Job arr[] = {{3, 10, 20}, {1, 2, 50}, {6, 19, 100}, {2, 100, 200}};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "The optimal profit is " << findMaxProfit(arr, n);
    return 0;
}

```

[Run on IDE](#)

Output:

The optimal profit is 250

Time Complexity of the above Dynamic Programming Solution is $O(n^2)$. Note that the above solution can be optimized to $O(n \log n)$ using Binary Search in `latestNonConflict()` instead of linear search. Thanks to Garvit for suggesting this optimization. Please refer below post for details.

[Weighted Job Scheduling in \$O\(n \log n\)\$ time](#)

References:

<http://courses.cs.washington.edu/courses/cse521/13wi/slides/06dp-sched.pdf>

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



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Dynamic Programming Dynamic Programming

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 33 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)
[GATE CS](#)
[Placements](#)
[GeeksQuiz](#)


[Login/Register](#)

Longest Even Length Substring such that Sum of First and Second Half is same

Given a string 'str' of digits, find length of the longest substring of 'str', such that the length of the substring is 2k digits and sum of left k digits is equal to the sum of right k digits.

Examples:

Input: str = "123123"

Output: 6

The complete string is of even length and sum of first and second half digits is same

Input: str = "1538023"

Output: 4

The longest substring with same first and second half sum is "5380"

We strongly recommend that you click here and practice it, before moving on to the solution.

Simple Solution [O(n³)]

A Simple Solution is to check every substring of even length. The following is C based implementation of simple approach.

```
// A simple C based program to find length of longest even length
// substring with same sum of digits in left and right
#include<stdio.h>
#include<string.h>

int findLength(char *str)
{
    int n = strlen(str);
    int maxlen = 0; // Initialize result

    // Choose starting point of every substring
    for (int i=0; i<n; i++)
    {
        // Choose ending point of even length substring
        for (int j =i+1; j<n; j += 2)
        {
            int length = j-i+1;//Find length of current substr
            int sum1 = 0, sum2 = 0;
            for (int k=0; k<length; k++)
                if (i+k < j)
                    sum1 += str[i+k] - '0';
                else
                    sum2 += str[j+k] - '0';

            if (sum1 == sum2)
                maxlen = length;
        }
    }
    return maxlen;
}
```

```

// Calculate left & right sums for current substr
int leftsum = 0, rightsum =0;
for (int k =0; k<length/2; k++)
{
    leftsum += (str[i+k]-'0');
    rightsum += (str[i+k+length/2]-'0');
}

// Update result if needed
if (leftsum == rightsum && maxlen < length)
    maxlen = length;
}

return maxlen;
}

// Driver program to test above function
int main(void)
{
    char str[] = "1538023";
    printf("Length of the substring is %d", findLength(str));
    return 0;
}

```

[Run on IDE](#)

Output:

Length of the substring is 4

Dynamic Programming [O(n²) and O(n²) extra space]

The above solution can be optimized to work in O(n²) using **Dynamic Programming**. The idea is to build a 2D table that stores sums of substrings. The following is C based implementation of Dynamic Programming approach.

```

// A C based program that uses Dynamic Programming to find length of the
// longest even substring with same sum of digits in left and right half
#include <stdio.h>
#include <string.h>

int findLength(char *str)
{
    int n = strlen(str);
    int maxlen = 0; // Initialize result

    // A 2D table where sum[i][j] stores sum of digits
    // from str[i] to str[j]. Only filled entries are
    // the entries where j >= i
    int sum[n][n];

    // Fill the diagonal values for substrings of length 1
    for (int i =0; i<n; i++)
        sum[i][i] = str[i]-'0';

    // Fill entries for substrings of length 2 to n
    for (int len=2; len<=n; len++)
    {
        // Pick i and j for current substring
        for (int i=0; i<n-len+1; i++)
        {
            int j = i+len-1;
            int k = len/2;

            // Calculate value of sum[i][j]
            sum[i][j] = sum[i][j-k] + sum[j-k+1][j];

            // Update result if 'len' is even, left and right
            // sums are same and len is more than maxlen
        }
    }
}

```

```

        if (len%2 == 0 && sum[i][j-k] == sum[(j-k+1)][j]
            && len > maxlen)
                maxlen = len;
    }
    return maxlen;
}

// Driver program to test above function
int main(void)
{
    char str[] = "153803";
    printf("Length of the substring is %d", findLength(str));
    return 0;
}

```

[Run on IDE](#)

Output:

Length of the substring is 4

Time complexity of the above solution is $O(n^2)$, but it requires $O(n^2)$ extra space.

[A $O(n^2)$ and $O(n)$ extra space solution]

The idea is to use a single dimensional array to store cumulative sum.

```

// A O(n^2) time and O(n) extra space solution
#include<bits/stdc++.h>
using namespace std;

int findLength(string str, int n)
{
    int sum[n+1]; // To store cumulative sum from first digit to nth digit
    sum[0] = 0;

    /* Store cumulative sum of digits from first to last digit */
    for (int i = 1; i <= n; i++)
        sum[i] = (sum[i-1] + str[i-1] - '0'); /* convert chars to int */

    int ans = 0; // initialize result

    /* consider all even length substrings one by one */
    for (int len = 2; len <= n; len += 2)
    {
        for (int i = 0; i <= n-len; i++)
        {
            int j = i + len - 1;

            /* Sum of first and second half is same than update ans */
            if (sum[i+len/2] - sum[i] == sum[i+len] - sum[i+len/2])
                ans = max(ans, len);
        }
    }
    return ans;
}

// Driver program to test above function
int main()
{
    string str = "123123";
    cout << "Length of the substring is " << findLength(str, str.length());
    return 0;
}

```

[Run on IDE](#)

Output:

Length of the substring is 6

Thanks to Gaurav Ahirwar for suggesting this method.

[A O(n^2) time and O(1) extra space solution]

The idea is to consider all possible mid points (of even length substrings) and keep expanding on both sides to get and update optimal length as the sum of two sides become equal.

Below is C++ implementation of the above idea.

```
// A O(n^2) time and O(1) extra space solution
#include<bits/stdc++.h>
using namespace std;

int findLength(string str, int n)
{
    int ans = 0; // Initialize result

    // Consider all possible midpoints one by one
    for (int i = 0; i <= n-2; i++)
    {
        /* For current midpoint 'i', keep expanding substring on
           both sides, if sum of both sides becomes equal update
           ans */
        int l = i, r = i + 1;

        /* initialize left and right sum */
        int lsum = 0, rsum = 0;

        /* move on both sides till indexes go out of bounds */
        while (r < n && l >= 0)
        {
            lsum += str[l] - '0';
            rsum += str[r] - '0';
            if (lsum == rsum)
                ans = max(ans, r-l+1);
            l--;
            r++;
        }
    }
    return ans;
}

// Driver program to test above function
int main()
{
    string str = "123123";
    cout << "Length of the substring is " << findLength(str, str.length());
    return 0;
}
```

Run on IDE

Output:

Length of the substring is 6

Thanks to Gaurav Ahirwar for suggesting this method.

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



GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [Dynamic Programming](#)

Related Posts:

- Minimum and Maximum values of an expression with * and +
- Number of palindromic paths in a matrix
- Maximum sum subarray removing at most one element
- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **39** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

HANDBOOK OF ALGORITHMS

Section
Pattern & String

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Searching for Patterns | Set 1 (Naive Pattern Searching)

Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. You may assume that $n > m$.

Examples:

```
Input: txt[] = "THIS IS A TEST TEXT"
      pat[] = "TEST"
Output: Pattern found at index 10
```

```
Input: txt[] = "AABAACAAADAABAABA"
      pat[] = "AABA"
Output: Pattern found at index 0
        Pattern found at index 9
        Pattern found at index 12
```

Text : **A A B A A C A A D A A B A A B A**

Pattern : **A A B A**

A A B A	A A B A
A A B A A C A A D A A B A A B A 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 A A B A	

Pattern Found at 1, 9 and 12

Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

We strongly recommend that you click [here](#) and practice it, before moving on to the solution.

Naive Pattern Searching:

Slide the pattern over text one by one and check for a match. If a match is found, then slides by 1 again to check for subsequent matches.

```
// C program for Naive Pattern Searching algorithm
#include<stdio.h>
#include<string.h>

void search(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++)
    {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i+j] != pat[j])
                break;

        if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            printf("Pattern found at index %d \n", i);
    }
}

/* Driver program to test above function */
int main()
{
    char txt[] = "AABAACAAADAABAAABAA";
    char pat[] = "AABA";
    search(pat, txt);
    return 0;
}
```

[Run on IDE](#)

Python

```
# Python program for Naive Pattern Searching
def search(pat, txt):
    M = len(pat)
    N = len(txt)

    # A loop to slide pat[] one by one
    for i in xrange(N-M+1):

        # For current index i, check for pattern match
        for j in xrange(M):
            if txt[i+j] != pat[j]:
                break
        if j == M-1: # if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            print "Pattern found at index " + str(i)

# Driver program to test the above function
txt = "AABAACAAADAABAAABAA"
pat = "AABA"
search (pat, txt)

# This code is contributed by Bhavya Jain
```

[Run on IDE](#)

Output:

```
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
```

What is the best case?

The best case occurs when the first character of the pattern is not present in text at all.

```
txt[] = "AABCCAADDEE"
pat[] = "FAA"
```

[Run on IDE](#)

The number of comparisons in best case is $O(n)$.

What is the worst case ?

The worst case of Naive Pattern Searching occurs in following scenarios.

- 1) When all characters of the text and pattern are same.

```
txt[] = "AAAAAAAAAAAAAAA"
pat[] = "AAAAA".
```

[Run on IDE](#)

- 2) Worst case also occurs when only the last character is different.

```
txt[] = "AAAAAAAAAAAAAAAB"
pat[] = "AAAAB".
```

[Run on IDE](#)

Number of comparisons in worst case is $O(m*(n-m+1))$. Although strings which have repeated characters are not likely to appear in English text, they may well occur in other applications (for example, in binary texts). The KMP matching algorithm improves the worst case to $O(n)$. We will be covering KMP in the next post. Also, we will be writing more posts to cover all pattern searching algorithms and data structures.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

[Pattern Searching](#)[Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

1.8

Average Difficulty : **1.8/5.0**
Based on **66** vote(s)

[Add to TODO List](#)[Mark as DONE](#)

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Searching for Patterns | Set 2 (KMP Algorithm)

Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. You may assume that $n > m$.

Examples:

```
Input: txt[] = "THIS IS A TEST TEXT"
       pat[] = "TEST"
```

Output: Pattern found at index 10

```
Input: txt[] = "AABAACAAADAABAABA"
       pat[] = "AABA"
```

Output: Pattern found at index 0

Pattern found at index 9

Pattern found at index 12

Text : **A A B A A C A A D A A B A A B A**

Pattern : **A A B A**



Pattern Found at 1, 9 and 12

We strongly recommend that you click here and practice it, before moving on to the solution.

Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

We have discussed Naive pattern searching algorithm in the [previous post](#). The worst case complexity of Naive algorithm is $O(m(n-m+1))$. Time complexity of KMP algorithm is $O(n)$ in worst case.

KMP (Knuth Morris Pratt) Pattern Searching

The [Naive pattern searching algorithm](#) doesn't work well in cases where we see many matching characters followed by a mismatching character. Following are some examples.

```

txt[] = "AAAAAAAAAAAAAAAB"
pat[] = "AAAAB"

txt[] = "ABABABCABABCABABCAB"
pat[] = "ABABAC" (not a worst case, but a bad case for Naive)

```

The KMP matching algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to O(n). The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text of next window. We take advantage of this information to avoid matching the characters that we know will anyway match. Let us consider below example to understand this.

Matching Overview

```

txt = "AAAAABAAABA"
pat = "AAAA"

```

We compare first window of **txt** with **pat**
txt = "AAAAABAAABA"
pat = "AAAA" [Initial position]
We find a match. This is same as **Naive String Matching**.

In the next step, we compare next window of **txt** with **pat**.

```

txt = "AAAAABAAABA"
pat = "AAAA" [Pattern shifted one position]

```

This is where KMP does optimization over Naive. In this second window, we only compare fourth **A** of pattern with fourth character of current window of text to decide whether current window matches or not. Since we know first three characters will anyway match, we skipped matching first three characters.

Need of Preprocessing?

An important question arises from above explanation, how to know how many characters to be skipped. To know this, we pre-process pattern and prepare an integer array **lps[]** that tells us count of characters to be skipped.

Preprocessing Overview:

- KMP algorithm does preprocesses **pat[]** and constructs an auxiliary **lps[]** of size m (same as size of pattern) which is used to skip characters while matching.
- **name lps indicates longest proper prefix which is also suffix..** A proper prefix is prefix with whole string **not** allowed. For example, prefixes of "ABC" are "", "A", "AB" and "ABC". Proper prefixes are "", "A" and "AB". Suffixes of the string are "", "C", "BC" and "ABC".
- For each sub-pattern **pat[0..i]** where i = 0 to m-1, **lps[i]** stores length of the maximum matching proper prefix which is also a suffix of the sub-pattern **pat[0..i]**.

```

lps[i] = the longest proper prefix of pat[0..i]
which is also a suffix of pat[0..i].

```

Note : **lps[i]** could also be defined as longest prefix which is also proper suffix. We need to use proper at one place to make sure that the whole substring is not considered.

Examples of lps[] construction:

For the pattern "AAAA",

lps[] is [0, 1, 2, 3]

For the pattern "ABCDE",

lps[] is [0, 0, 0, 0, 0]

For the pattern "AABAACAAABAA",

lps[] is [0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5]

For the pattern "AAACAAAAAC",

lps[] is [0, 1, 2, 0, 1, 2, 3, 3, 3, 4]

For the pattern "AAABAAA",

lps[] is [0, 1, 2, 0, 1, 2, 3]

Searching Algorithm:

Unlike **Naive algorithm**, where we slide the pattern by one and compare all characters at each shift, we use a value from lps[] to decide the next characters to be matched. The idea is to not match character that we know will anyway match.

How to use lps[] to decide next positions (or to know number of characters to be skipped)?

- We start comparison of pat[j] with j = 0 with characters of current window of text.
- We keep matching characters txt[i] and pat[j] and keep incrementing i and j while pat[j] and txt[i] keep **matching**.
- When we see a **mismatch**
 - We know that characters pat[0..j-1] match with txt[i-j+1...i-1] (Note that j starts with 0 and increment it only when there is a match).
 - We also know (from above definition) that lps[j-1] is count of characters of pat[0...j-1] that are both proper prefix and suffix.
 - From above two points, we can conclude that we do not need to match these lps[j-1] characters with txt[i-j...i-1] because we know that these characters will anyway match. Let us consider above example to understand this.

```
txt[] = "AAAAABAAABA"
```

```
pat[] = "AAAA"
```

```
lps[] = {0, 1, 2, 3}
```

```
i = 0, j = 0
```

```
txt[] = "AAAAABAAABA"
```

```
pat[] = "AAAA"
```

```
txt[i] and pat[j] match, do i++, j++
```

```
i = 1, j = 1
```

```
txt[] = "AAAAABAAABA"
```

```
pat[] = "AAAA"
```

```
txt[i] and pat[j] match, do i++, j++
```

```
i = 2, j = 2
```

```
txt[] = "AAAAABAAABA"
```

```
pat[] = "AAAA"
```

```
txt[i] and pat[j] match, do i++, j++
```

```
i = 3, j = 3
```

```

txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] match, do i++, j++

i = 4, j = 4
Since j == M, print pattern found and reset j,
j = lps[j-1] = lps[3] = 3

```

Here unlike Naive algorithm, we do not match first three characters of this window. Value of $lps[j-1]$ (in above step) gave us index of next character to match.

```

i = 4, j = 3
txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] match, do i++, j++

```

```

i = 5, j = 4
Since j == M, print pattern found and reset j,
j = lps[j-1] = lps[3] = 3

```

Again unlike Naive algorithm, we do not match first three characters of this window. Value of $lps[j-1]$ (in above step) gave us index of next character to match.

```

i = 5, j = 3
txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] do NOT match and j > 0, change only j
j = lps[j-1] = lps[2] = 2

```

```

i = 5, j = 2
txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] do NOT match and j > 0, change only j
j = lps[j-1] = lps[1] = 1

```

```

i = 5, j = 1
txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] do NOT match and j > 0, change only j
j = lps[j-1] = lps[0] = 0

```

```

i = 5, j = 0
txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] do NOT match and j is 0, we do i++.

```

```

i = 6, j = 0
txt[] = "AAAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] match, do i++ and j++

```

```

i = 7, j = 1
txt[] = "AAAAABAAABA"
pat[] = "AAAAA"
txt[i] and pat[j] match, do i++ and j++

```

We continue this way...

```

// C++ program for implementation of KMP pattern searching
// algorithm
#include<bits/stdc++.h>

void computeLPSArray(char *pat, int M, int *lps);

// Prints occurrences of txt[] in pat[]
void KMPSearch(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    // create lps[] that will hold the longest prefix suffix
    // values for pattern
    int lps[M];

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]
    int j = 0; // index for pat[]
    while (i < N)
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }

        if (j == M)
        {
            printf("Found pattern at index %d \n", i-j);
            j = lps[j-1];
        }

        // mismatch after j matches
        else if (i < N && pat[j] != txt[i])
        {
            // Do not match lps[0..lps[j-1]] characters,
            // they will match anyway
            if (j != 0)
                j = lps[j-1];
            else
                i = i+1;
        }
    }
}

// Fills lps[] for given pattern pat[0..M-1]
void computeLPSArray(char *pat, int M, int *lps)
{
    // length of the previous longest prefix suffix
    int len = 0;

    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    int i = 1;
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            // This is tricky. Consider the example.
            // AACAAAAA and i = 7. The idea is similar
            lps[i] = 0;
            i++;
        }
    }
}

```

```

// to search step.
if (len != 0)
{
    len = lps[len-1];

    // Also, note that we do not increment
    // i here
}
else // if (len == 0)
{
    lps[i] = 0;
    i++;
}
}

// Driver program to test above function
int main()
{
    char *txt = "ABABDABACDABABCABAB";
    char *pat = "ABABCABAB";
    KMPSearch(pat, txt);
    return 0;
}

```

[Run on IDE](#)

Java

```

// JAVA program for implementation of KMP pattern
// searching algorithm

class KMP_String_Matching
{
    void KMPSearch(String pat, String txt)
    {
        int M = pat.length();
        int N = txt.length();

        // create lps[] that will hold the longest
        // prefix suffix values for pattern
        int lps[] = new int[M];
        int j = 0; // index for pat[]

        // Preprocess the pattern (calculate lps[]
        // array)
        computeLPSArray(pat,M,lps);

        int i = 0; // index for txt[]
        while (i < N)
        {
            if (pat.charAt(j) == txt.charAt(i))
            {
                j++;
                i++;
            }
            if (j == M)
            {
                System.out.println("Found pattern "+
                    "at index " + (i-j));
                j = lps[j-1];
            }

            // mismatch after j matches
            else if (i < N && pat.charAt(j) != txt.charAt(i))
            {
                // Do not match lps[0..lps[j-1]] characters,
                // they will match anyway
                if (j != 0)
                    j = lps[j-1];
            }
        }
    }
}

```

```

        else
            i = i+1;
    }

void computeLPSArray(String pat, int M, int lps[])
{
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;
    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M)
    {
        if (pat.charAt(i) == pat.charAt(len))
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            // This is tricky. Consider the example.
            // AAACAAAA and i = 7. The idea is similar
            // to search step.
            if (len != 0)
            {
                len = lps[len-1];

                // Also, note that we do not increment
                // i here
            }
            else // if (len == 0)
            {
                lps[i] = len;
                i++;
            }
        }
    }
}

// Driver program to test above function
public static void main(String args[])
{
    String txt = "ABABDABACDABABCABAB";
    String pat = "ABABCABAB";
    new KMP_String_Matching().KMPSearch(pat,txt);
}
// This code has been contributed by Amit Khandelwal.

```

[Run on IDE](#)

Python

```

# Python program for KMP Algorithm
def KMPSearch(pat, txt):
    M = len(pat)
    N = len(txt)

    # create lps[] that will hold the longest prefix suffix
    # values for pattern
    lps = [0]*M
    j = 0 # index for pat[]

    # Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps)

    i = 0 # index for txt[]

```

```

while i < N:
    if pat[j] == txt[i]:
        i += 1
        j += 1

    if j == M:
        print "Found pattern at index " + str(i-j)
        j = lps[j-1]

    # mismatch after j matches
    elif i < N and pat[j] != txt[i]:
        # Do not match lps[0..lps[j-1]] characters,
        # they will match anyway
        if j != 0:
            j = lps[j-1]
        else:
            i += 1

def computeLPSArray(pat, M, lps):
    len = 0 # length of the previous longest prefix suffix

    lps[0] # lps[0] is always 0
    i = 1

    # the loop calculates lps[i] for i = 1 to M-1
    while i < M:
        if pat[i]==pat[len]:
            len += 1
            lps[i] = len
            i += 1
        else:
            # This is tricky. Consider the example.
            # AACAAAAA and i = 7. The idea is similar
            # to search step.
            if len != 0:
                len = lps[len-1]

            # Also, note that we do not increment i here
        else:
            lps[i] = 0
            i += 1

txt = "ABABDABACDABABCABAB"
pat = "ABABCABAB"
KMPSearch(pat, txt)

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

```
Found pattern at index 10
```

Preprocessing Algorithm:

In the preprocessing part, we calculate values in `lps[]`. To do that, we keep track of the length of the longest prefix suffix value (we use `len` variable for this purpose) for the previous index. We initialize `lps[0]` and `len` as 0. If `pat[len]` and `pat[i]` match, we increment `len` by 1 and assign the incremented value to `lps[i]`. If `pat[i]` and `pat[len]` do not match and `len` is not 0, we update `len` to `lps[len-1]`. See `computeLPSArray ()` in the below code for details.

Illustration of preprocessing (or construction of `lps[]`)

```

pat[] = "AACAAAAA"

len = 0, i = 0.

```

```
lps[0] is always 0, we move  
to i = 1

len = 0, i = 1.  
Since pat[len] and pat[i] match, do len++,  
store it in lps[i] and do i++.  
len = 1, lps[1] = 1, i = 2

len = 1, i = 2.  
Since pat[len] and pat[i] match, do len++,  
store it in lps[i] and do i++.  
len = 2, lps[2] = 2, i = 3

len = 2, i = 3.  
Since pat[len] and pat[i] do not match, and len > 0,  
set len = lps[len-1] = lps[1] = 1

len = 1, i = 3.  
Since pat[len] and pat[i] do not match and len > 0,  
len = lps[len-1] = lps[0] = 0

len = 0, i = 3.  
Since pat[len] and pat[i] do not match and len = 0,  
Set lps[3] = 0 and i = 4.

len = 0, i = 4.  
Since pat[len] and pat[i] match, do len++,  
store it in lps[i] and do i++.  
len = 1, lps[4] = 1, i = 5

len = 1, i = 5.  
Since pat[len] and pat[i] match, do len++,  
store it in lps[i] and do i++.  
len = 2, lps[5] = 2, i = 6

len = 2, i = 6.  
Since pat[len] and pat[i] match, do len++,  
store it in lps[i] and do i++.  
len = 3, lps[6] = 3, i = 7

len = 3, i = 7.  
Since pat[len] and pat[i] do not match and len > 0,  
set len = lps[len-1] = lps[2] = 2

len = 2, i = 7.  
Since pat[len] and pat[i] match, do len++,  
store it in lps[i] and do i++.  
len = 3, lps[7] = 3, i = 8
```

We stop here as we have constructed the whole lps[].

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

4.2

Average Difficulty : 4.2/5.0
Based on 136 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Searching for Patterns | Set 3 (Rabin-Karp Algorithm)

Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. You may assume that $n > m$.

Examples:

```
Input: txt[] = "THIS IS A TEST TEXT"
       pat[] = "TEST"
```

```
Output: Pattern found at index 10
```

```
Input: txt[] = "AABAACAAADAABAABA"
       pat[] = "AABA"
```

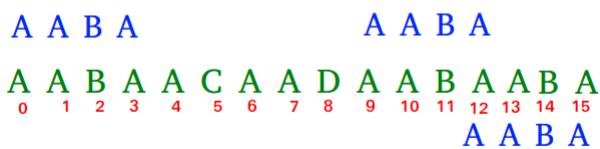
```
Output: Pattern found at index 0
```

```
Pattern found at index 9
```

```
Pattern found at index 12
```

Text : **A A B A A C A A D A A B A A B A**

Pattern : **A A B A**



Pattern Found at 1, 9 and 12

The [Naive String Matching](#) algorithm slides the pattern one by one. After each slide, it one by one checks characters at the current shift and if all characters match then prints the match.

Like the Naive Algorithm, Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, Rabin Karp algorithm matches the hash value of the pattern with the hash value of current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for following strings.

- 1) Pattern itself.
- 2) All the substrings of text of length m.

Since we need to efficiently calculate hash values for all the substrings of size m of text, we must have a hash function which has following property.

Hash at the next shift must be efficiently computable from the current hash value and next character in text or we

can say $\text{hash}(\text{txt}[s+1 \dots s+m])$ must be efficiently computable from $\text{hash}(\text{txt}[s \dots s+m-1])$ and $\text{txt}[s+m]$ i.e., $\text{hash}(\text{txt}[s+1 \dots s+m]) = \text{rehash}(\text{txt}[s+m], \text{hash}(\text{txt}[s \dots s+m-1]))$ and rehash must be O(1) operation.

The hash function suggested by Rabin and Karp calculates an integer value. The integer value for a string is numeric value of a string. For example, if all possible characters are from 1 to 10, the numeric value of "122" will be 122. The number of possible characters is higher than 10 (256 in general) and pattern length can be large. So the numeric values cannot be practically stored as an integer. Therefore, the numeric value is calculated using modular arithmetic to make sure that the hash values can be stored in an integer variable (can fit in memory words). To do rehashing, we need to take off the most significant digit and add the new least significant digit for in hash value. Rehashing is done using the following formula.

$$\text{hash}(\text{txt}[s+1 \dots s+m]) = d (\text{hash}(\text{txt}[s \dots s+m-1]) - \text{txt}[s]*h) + \text{txt}[s+m] \bmod q$$

$\text{hash}(\text{txt}[s \dots s+m-1])$: Hash value at shift s.

$\text{hash}(\text{txt}[s+1 \dots s+m])$: Hash value at next shift (or shift $s+1$)

d: Number of characters in the alphabet

q: A prime number

h: d^{m-1}

```
/* Following program is a C implementation of Rabin Karp
Algorithm given in the CLRS book */
#include<stdio.h>
#include<string.h>
```

```
// d is the number of characters in input alphabet
#define d 256
```

```
/* pat -> pattern
   txt -> text
   q -> A prime number
*/
void search(char pat[], char txt[], int q)
```

```
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0; // hash value for pattern
    int t = 0; // hash value for txt
    int h = 1;

    // The value of h would be "pow(d, M-1)%q"
    for (i = 0; i < M-1; i++)
        h = (h*d)%q;

    // Calculate the hash value of pattern and first
    // window of text
    for (i = 0; i < M; i++)
    {
        p = (d*p + pat[i])%q;
        t = (d*t + txt[i])%q;
    }

    // Slide the pattern over text one by one
    for (i = 0; i <= N - M; i++)
    {

        // Check the hash values of current window of text
        // and pattern. If the hash values match then only
        // check for characters one by one
        if ( p == t )
        {
            /* Check for characters one by one */
            for (j = 0; j < M; j++)

```

```

    {
        if (txt[i+j] != pat[j])
            break;
    }

    // if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
    if (j == M)
        printf("Pattern found at index %d \n", i);
}

// Calculate hash value for next window of text: Remove
// leading digit, add trailing digit
if ( i < N-M )
{
    t = (d*(t - txt[i]*h) + txt[i+M])%q;

    // We might get negative value of t, converting it
    // to positive
    if (t < 0)
        t = (t + q);
}
}

/* Driver program to test above function */
int main()
{
    char txt[] = "GEEKS FOR GEEKS";
    char pat[] = "GEEK";
    int q = 101; // A prime number
    search(pat, txt, q);
    return 0;
}

```

[Run on IDE](#)

Python

```

# Following program is the python implementation of
# Rabin Karp Algorithm given in CLRS book

# d is the number of characters in input alphabet
d = 256

# pat  -> pattern
# txt  -> text
# q     -> A prime number

def search(pat, txt, q):
    M = len(pat)
    N = len(txt)
    i = 0
    j = 0
    p = 0      # hash value for pattern
    t = 0      # hash value for txt
    h = 1

    # The value of h would be "pow(d, M-1)%q"
    for i in xrange(M-1):
        h = (h*d)%q

    # Calculate the hash value of pattern and first window
    # of text
    for i in xrange(M):
        p = (d*p + ord(pat[i]))%q
        t = (d*t + ord(txt[i]))%q

    # Slide the pattern over text one by one
    for i in xrange(N-M+1):
        # Check the hash values of current window of text and
        # pattern if the hash values match then only check

```

```

# for characters on by one
if p==t:
    # Check for characters one by one
    for j in xrange(M):
        if txt[i+j] != pat[j]:
            break

    j+=1
    # if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
    if j==M:
        print "Pattern found at index " + str(i)

# Calculate hash value for next window of text: Remove
# leading digit, add trailing digit
if i < N-M:
    t = (d*(t-ord(txt[i])*h) + ord(txt[i+M]))%q

    # We might get negative values of t, converting it to
    # positive
    if t < 0:
        t = t+q

# Driver program to test the above function
txt = "GEEKS FOR GEEKS"
pat = "GEEK"
q = 101 # A prime number
search(pat,txt,q)

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

```

Pattern found at index 0
Pattern found at index 10

```

The average and best case running time of the Rabin-Karp algorithm is $O(n+m)$, but its worst-case time is $O(nm)$. Worst case of Rabin-Karp algorithm occurs when all characters of pattern and text are same as the hash values of all the substrings of $txt[]$ match with hash value of $pat[]$. For example $pat[] = "AAA"$ and $txt[] = "AAAAAAA"$.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

<http://net.pku.edu.cn/~course/cs101/2007/resource/Intro2Algorithm/book6/chap34.htm>

<http://www.cs.princeton.edu/courses/archive/fall04/cos226/lectures/string.4up.pdf>

http://en.wikipedia.org/wiki/Rabin-Karp_string_search_algorithm

Related Posts:

[Searching for Patterns | Set 1 \(Naive Pattern Searching\)](#)

[Searching for Patterns | Set 2 \(KMP Algorithm\)](#)



Free Download

Zip, Unzip or Open Any File.

Unzipper



GATE CS Corner Company Wise Coding Practice

Pattern Searching modular-arithmetic Pattern Searching

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 34 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Searching for Patterns | Set 4 (A Naive Pattern Searching Question)

Question: We have discussed Naive String matching algorithm [here](#). Consider a situation where all characters of pattern are different. Can we modify the [original Naive String Matching algorithm](#) so that it works better for these types of patterns. If we can, then what are the changes to original algorithm?

Solution: In the [original Naive String matching algorithm](#), we always slide the pattern by 1. When all characters of pattern are different, we can slide the pattern by more than 1. Let us see how can we do this. When a mismatch occurs after j matches, we know that the first character of pattern will not match the j matched characters because all characters of pattern are different. So we can always slide the pattern by j without missing any valid shifts. Following is the modified code that is optimized for the special patterns.

```
/* C program for A modified Naive Pattern Searching
   algorithm that is optimized for the cases when all
   characters of pattern are different */
#include<stdio.h>
#include<string.h>

/* A modified Naive Pettern Searching algorithn that is optimized
   for the cases when all characters of pattern are different */
void search(char pat[], char txt[])
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i = 0;

    while (i <= N - M)
    {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i+j] != pat[j])
                break;

        if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
        {
            printf("Pattern found at index %d \n", i);
            i = i + M;
        }
        else if (j == 0)
            i = i + 1;
        else
            i = i + j; // slide the pattern by j
    }
}
```

```
/* Driver program to test above function */
int main()
{
    char txt[] = "ABCEABCDEABC";
    char pat[] = "ABCD";
    search(pat, txt);
    return 0;
}
```

[Run on IDE](#)

Python

```
# Python program for A modified Naive Pattern Searching
# algorithm that is optimized for the cases when all
# characters of pattern are different
def search(pat, txt):
    M = len(pat)
    N = len(txt)
    i = 0

    while i <= N-M:
        # For current index i, check for pattern match
        for j in xrange(M):
            if txt[i+j] != pat[j]:
                break
            j += 1

        if j==M:      # if pat[0...M-1] = txt[i,i+1,...i+M-1]
            print "Pattern found at index " + str(i)
            i = i + M
        elif j==0:
            i = i + 1
        else:
            i = i+ j    # slide the pattern by j

# Driver program to test the above function
txt = "ABCEABCDEABC"
pat = "ABCD"
search(pat, txt)

# This code is contributed by Bhavya Jain
```

[Run on IDE](#)

Output:

```
Pattern found at index 4
Pattern found at index 12
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Pattern Searching Pattern Searching

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

2

Average Difficulty : 2/5.0
Based on 19 vote(s)



Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Searching for Patterns | Set 5 (Finite Automata)

Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. You may assume that $n > m$.

Examples:

```
Input: txt[] = "THIS IS A TEST TEXT"
       pat[] = "TEST"
```

```
Output: Pattern found at index 10
```

```
Input: txt[] = "AABAACAAADAABAABA"
       pat[] = "AABA"
```

```
Output: Pattern found at index 0
```

```
Pattern found at index 9
```

```
Pattern found at index 12
```

Text : **A A B A A C A A D A A B A A B A**

Pattern : **A A B A**



Pattern Found at 1, 9 and 12

Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

We have discussed the following algorithms in the previous posts:

[Naive Algorithm](#)

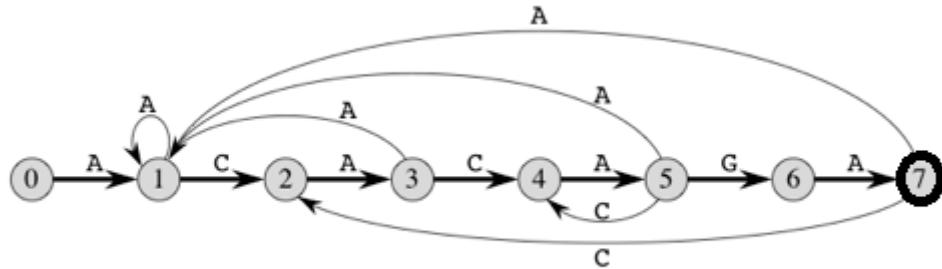
[KMP Algorithm](#)

[Rabin Karp Algorithm](#)

In this post, we will discuss Finite Automata (FA) based pattern searching algorithm. In FA based algorithm, we preprocess the pattern and build a 2D array that represents a Finite Automata. Construction of the FA is the main tricky part of this algorithm. Once the FA is built, the searching is simple. In search, we simply need to start from the first state of the automata and the first character of the text. At every step, we consider next character of text, look for the next state in the built FA and move to a new state. If we reach the final state, then the pattern is found

in the text. The time complexity of the search process is $O(n)$.

Before we discuss FA construction, let us take a look at the following FA for pattern ACACAGA.



state	character			
	A	C	G	T
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	7	0	0	0
7	1	2	0	0

The above diagrams represent graphical and tabular representations of pattern ACACAGA.

Number of states in FA will be $M+1$ where M is length of the pattern. The main thing to construct FA is to get the next state from the current state for every possible character. Given a character x and a state k , we can get the next state by considering the string “ $\text{pat}[0..k-1]x$ ” which is basically concatenation of pattern characters $\text{pat}[0]$, $\text{pat}[1] \dots \text{pat}[k-1]$ and the character x . The idea is to get length of the longest prefix of the given pattern such that the prefix is also suffix of “ $\text{pat}[0..k-1]x$ ”. The value of length gives us the next state. For example, let us see how to get the next state from current state 5 and character ‘C’ in the above diagram. We need to consider the string, “ $\text{pat}[0..5]C$ ” which is “ACACAC”. The length of the longest prefix of the pattern such that the prefix is suffix of “ACACAC” is 4 (“ACAC”). So the next state (from state 5) is 4 for character ‘C’.

In the following code, `computeTF()` constructs the FA. The time complexity of the `computeTF()` is $O(m^3 \cdot \text{NO_OF_CHARS})$ where m is length of the pattern and NO_OF_CHARS is size of alphabet (total number of possible characters in pattern and text). The implementation tries all possible prefixes starting from the longest possible that can be a suffix of “ $\text{pat}[0..k-1]x$ ”. There are better implementations to construct FA in $O(m \cdot \text{NO_OF_CHARS})$ (Hint: we can use something like [LPS array construction in KMP algorithm](#)). We have covered the better implementation in our [next post on pattern searching](#).

```
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

int getNextState(char *pat, int M, int state, int x)
{
    // If the character c is same as next character in pattern,
    // then simply increment state
    if (state < M && x == pat[state])
        return state+1;

    int ns, i; // ns stores the result which is next state

    // ns finally contains the longest prefix which is also suffix
    // in "pat[0..state-1]c"
}
```

```

// Start from the largest possible value and stop when you find
// a prefix which is also suffix
for (ns = state; ns > 0; ns--)
{
if(pat[ns-1] == x)
{
for(i = 0; i < ns-1; i++)
{
if (pat[i] != pat[state-ns+1+i])
break;
}
if (i == ns-1)
return ns;
}
}

return 0;
}

/* This function builds the TF table which represents Finite Automata for a
given pattern */
void computeTF(char *pat, int M, int TF[][NO_OF_CHARS])
{
int state, x;
for (state = 0; state <= M; ++state)
for (x = 0; x < NO_OF_CHARS; ++x)
TF[state][x] = getNextState(pat, M, state, x);
}

/* Prints all occurrences of pat in txt */
void search(char *pat, char *txt)
{
int M = strlen(pat);
int N = strlen(txt);

int TF[M+1][NO_OF_CHARS];

computeTF(pat, M, TF);

// Process txt over FA.
int i, state=0;
for (i = 0; i < N; i++)
{
state = TF[state][txt[i]];
if (state == M)
{
printf ("\n Pattern found at index %d", i-M+1);
}
}
}

// Driver program to test above function
int main()
{
char *txt = "AABAACAAADAABAAABAA";
char *pat = "AABA";
search(pat, txt);
return 0;
}

```

[Run on IDE](#)

Output:

```

Pattern found at index 0
Pattern found at index 9
Pattern found at index 13

```

References:

Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#)
[Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

4.3

Average Difficulty : 4.3/5.0
Based on 11 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

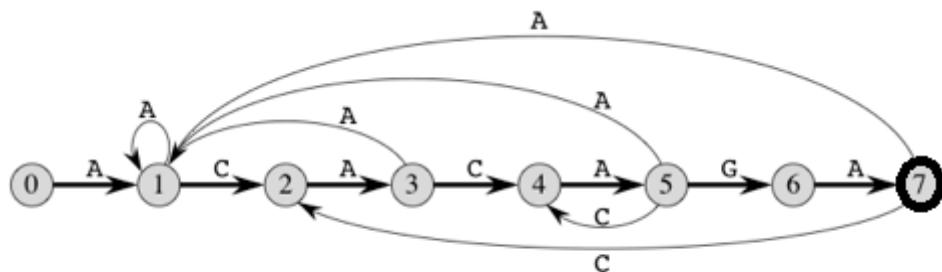
Google™ Custom Search



[Login/Register](#)

Pattern Searching | Set 6 (Efficient Construction of Finite Automata)

In the [previous post](#), we discussed Finite Automata based pattern searching algorithm. The FA (Finite Automata) construction method discussed in previous post takes $O((m^3)*NO_OF_CHARS)$ time. FA can be constructed in $O(m*NO_OF_CHARS)$ time. In this post, we will discuss the $O(m*NO_OF_CHARS)$ algorithm for FA construction. The idea is similar to Ips (longest prefix suffix) array construction discussed in the [KMP algorithm](#). We use previously filled rows to fill a new row.



state	character			
	A	C	G	T
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	7	0	0	0
7	1	2	0	0

The above diagrams represent graphical and tabular representations of pattern ACACAGA.

Algorithm:

- 1) Fill the first row. All entries in first row are always 0 except the entry for pat[0] character. For pat[0] character, we always need to go to state 1.
- 2) Initialize Ips as 0. Ips for the first index is always 0.
- 3) Do following for rows at index i = 1 to M. (M is the length of the pattern)
 -a) Copy the entries from the row at index equal to Ips.
 -b) Update the entry for pat[i] character to i+1.
 -c) Update Ips “Ips = TF[Ips][pat[i]]” where TF is the 2D array which is being constructed.

Implementation

Following is C implementation for the above algorithm.

```
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

/* This function builds the TF table which represents Finite Automata for a
   given pattern */
void computeTransFun(char *pat, int M, int TF[][NO_OF_CHARS])
{
    int i, lps = 0, x;

    // Fill entries in first row
    for (x = 0; x < NO_OF_CHARS; x++)
        TF[0][x] = 0;
    TF[0][pat[0]] = 1;

    // Fill entries in other rows
    for (i = 1; i <= M; i++)
    {
        // Copy values from row at index lps
        for (x = 0; x < NO_OF_CHARS; x++)
            TF[i][x] = TF[lps][x];

        // Update the entry corresponding to this character
        TF[i][pat[i]] = i + 1;

        // Update lps for next row to be filled
        if (i < M)
            lps = TF[lps][pat[i]];
    }
}

/* Prints all occurrences of pat in txt */
void search(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    int TF[M+1][NO_OF_CHARS];

    computeTransFun(pat, M, TF);

    // process text over FA.
    int i, j=0;
    for (i = 0; i < N; i++)
    {
        j = TF[j][txt[i]];
        if (j == M)
        {
            printf ("\n pattern found at index %d", i-M+1);
        }
    }
}

/* Driver program to test above function */
int main()
{
    char *txt = "GEEKS FOR GEEKS";
    char *pat = "GEEKS";
    search(pat, txt);
    getchar();
    return 0;
}
```

Run on IDE

Output:

```
pattern found at index 0
pattern found at index 10
```

Time Complexity for FA construction is $O(M \cdot NO_OF_CHARS)$. The code for search is same as the [previous post](#) and time complexity for it is $O(n)$.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Download

Free Download

www.unzipper.com



GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

3.5

Average Difficulty : 3.5/5.0
Based on 4 vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Pattern Searching | Set 7 (Boyer Moore Algorithm – Bad Character Heuristic)

Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. You may assume that $n > m$.

Examples:

```
Input: txt[] = "THIS IS A TEST TEXT"
      pat[] = "TEST"
Output: Pattern found at index 10
```

```
Input: txt[] = "AABAACAAADAABAABA"
      pat[] = "AABA"
Output: Pattern found at index 0
        Pattern found at index 9
        Pattern found at index 12
```

Text : **A A B A A C A A D A A B A A B A**

Pattern : **A A B A**

A A B A	A A B A
A A B A A C A A D A A B A A B A 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	A A B A

Pattern Found at 1, 9 and 12

Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

We have discussed the following algorithms in the previous posts:

[Naive Algorithm](#)

[KMP Algorithm](#)

[Rabin Karp Algorithm](#)

[Finite Automata based Algorithm](#)

In this post, we will discuss Boyer Moore pattern searching algorithm. Like [KMP](#) and [Finite Automata](#) algorithms, Boyer Moore algorithm also preprocesses the pattern.

[Boyer Moore is a combination of following two approaches.](#)

1) Bad Character Heuristic

2) Good Suffix Heuristic

Both of the above heuristics can also be used independently to search a pattern in a text. Let us first understand how two independent approaches work together in the Boyer Moore algorithm. If we take a look at the [Naive algorithm](#), it slides the pattern over the text one by one. KMP algorithm does preprocessing over the pattern so that the pattern can be shifted by more than one. The Boyer Moore algorithm does preprocessing for the same reason. It preprocesses the pattern and creates different arrays for both heuristics. At every step, it slides the pattern by max of the slides suggested by the two heuristics. [So it uses best of the two heuristics at every step. Unlike the previous pattern searching algorithms, Boyer Moore algorithm starts matching from the last character of the pattern.](#)

In this post, we will discuss bad character heuristic, and discuss Good Suffix heuristic in the next post.

The idea of bad character heuristic is simple. The character of the text which doesn't match with the current character of pattern is called the Bad Character. Whenever a character doesn't match, we slide the pattern in such a way that aligns the bad character with the last occurrence of it in pattern. We preprocess the pattern and store the last occurrence of every possible character in an array of size equal to alphabet size. If the character is not present at all, then it may result in a shift by m (length of pattern). Therefore, [the bad character heuristic takes \$O\(n/m\)\$ time in the best case.](#)

```
/* Program for Bad Character Heuristic of Boyer Moore String Matching Algorithm */
```

```
# include <limits.h>
# include <string.h>
# include <stdio.h>

# define NO_OF_CHARS 256

// A utility function to get maximum of two integers
int max (int a, int b) { return (a > b)? a: b; }

// The preprocessing function for Boyer Moore's bad character heuristic
void badCharHeuristic( char *str, int size, int badchar[NO_OF_CHARS])
{
    int i;

    // Initialize all occurrences as -1
    for (i = 0; i < NO_OF_CHARS; i++)
        badchar[i] = -1;

    // Fill the actual value of last occurrence of a character
    for (i = 0; i < size; i++)
        badchar[(int) str[i]] = i;
}

/* A pattern searching function that uses Bad Character Heuristic of
   Boyer Moore Algorithm */
void search( char *txt,  char *pat)
{
    int m = strlen(pat);
    int n = strlen(txt);

    int badchar[NO_OF_CHARS];

    /* Fill the bad character array by calling the preprocessing
       function badCharHeuristic() for given pattern */
    badCharHeuristic(pat, m, badchar);

    int s = 0; // s is shift of the pattern with respect to text
    while(s <= (n - m))
    {
        int j = m-1;

        /* Keep reducing index j of pattern while characters of
           pattern and text are matching at this shift s */

```

```

while(j >= 0 && pat[j] == txt[s+j])
    j--;

/* If the pattern is present at current shift, then index j
   will become -1 after the above loop */
if (j < 0)
{
    printf("\n pattern occurs at shift = %d", s);

    /* Shift the pattern so that the next character in text
       aligns with the last occurrence of it in pattern.
       The condition s+m < n is necessary for the case when
       pattern occurs at the end of text */
    s += (s+m < n)? m-badchar[txt[s+m]] : 1;

}

else
    /* Shift the pattern so that the bad character in text
       aligns with the last occurrence of it in pattern. The
       max function is used to make sure that we get a positive
       shift. We may get a negative shift if the last occurrence
       of bad character in pattern is on the right side of the
       current character. */
    s += max(1, j - badchar[txt[s+j]]);

}

/* Driver program to test above funtion */
int main()
{
    char txt[] = "ABAAABCD";
    char pat[] = "ABC";
    search(txt, pat);
    return 0;
}

```

[Run on IDE](#)

Output:

pattern occurs at shift = 4

The Bad Character Heuristic may take $O(mn)$ time in worst case. The worst case occurs when all characters of the text and pattern are same. For example, $\text{txt}[] = \text{"AAAAA.....AAAAA"}$ and $\text{pat}[] = \text{"AAAAA"}$.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

4.3

Average Difficulty : **4.3/5.0**
Based on **10** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Suffix Array | Set 1 (Introduction)

We strongly recommend to read following post on suffix trees as a pre-requisite for this post.

[Pattern Searching | Set 8 \(Suffix Tree Introduction\)](#)

A suffix array is a sorted array of all suffixes of a given string. The definition is similar to Suffix Tree which is compressed trie of all suffixes of the given text. Any suffix tree based algorithm can be replaced with an algorithm that uses a suffix array enhanced with additional information and solves the same problem in the same time complexity (Source [Wiki](#)).

A suffix array can be constructed from Suffix tree by doing a DFS traversal of the suffix tree. In fact Suffix array and suffix tree both can be constructed from each other in linear time.

Advantages of suffix arrays over suffix trees include improved space requirements, simpler linear time construction algorithms (e.g., compared to Ukkonen's algorithm) and improved cache locality (Source: [Wiki](#))

Example:

Let the given string be "banana".

0 banana		5 a
1 anana	Sort the Suffixes	3 ana
2 nana	----->	1 anana
3 ana	alphabetically	0 banana
4 na		4 na
5 a		2 nana

So the suffix array for "banana" is {5, 3, 1, 0, 4, 2}

Naive method to build Suffix Array

A simple method to construct suffix array is to make an array of all suffixes and then sort the array. Following is implementation of simple method.

```
// Naive algorithm for building suffix array of a given text
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

// Structure to store information of a suffix
struct suffix
{
    int index;
    char *suff;
};

// A comparison function used by sort() to compare two suffixes
```

```

int cmp(struct suffix a, struct suffix b)
{
    return strcmp(a.suff, b.suff) < 0? 1 : 0;
}

// This is the main function that takes a string 'txt' of size n as an
// argument, builds and return the suffix array for the given string
int *buildSuffixArray(char *txt, int n)
{
    // A structure to store suffixes and their indexes
    struct suffix suffixes[n];

    // Store suffixes and their indexes in an array of structures.
    // The structure is needed to sort the suffixes alphabetically
    // and maintain their old indexes while sorting
    for (int i = 0; i < n; i++)
    {
        suffixes[i].index = i;
        suffixes[i].suff = (txt+i);
    }

    // Sort the suffixes using the comparison function
    // defined above.
    sort(suffixes, suffixes+n, cmp);

    // Store indexes of all sorted suffixes in the suffix array
    int *suffixArr = new int[n];
    for (int i = 0; i < n; i++)
        suffixArr[i] = suffixes[i].index;

    // Return the suffix array
    return suffixArr;
}

// A utility function to print an array of given size
void printArr(int arr[], int n)
{
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    char txt[] = "banana";
    int n = strlen(txt);
    int *suffixArr = buildSuffixArray(txt, n);
    cout << "Following is suffix array for " << txt << endl;
    printArr(suffixArr, n);
    return 0;
}

```

Run on IDE

Output:

```

Following is suffix array for banana
5 3 1 0 4 2

```

The time complexity of above method to build suffix array is $O(n^2 \log n)$ if we consider a $O(n \log n)$ algorithm used for sorting. The sorting step itself takes $O(n^2 \log n)$ time as every comparison is a comparison of two strings and the comparison takes $O(n)$ time.

There are many efficient algorithms to build suffix array. We will soon be covering them as separate posts.

Search a pattern using the built Suffix Array

To search a pattern in a text, we preprocess the text and build a suffix array of the text. Since we have a sorted

array of all suffixes, [Binary Search](#) can be used to search. Following is the search function. Note that the function doesn't report all occurrences of pattern, it only report one of them.

```
// This code only contains search() and main. To make it a complete running
// above code or see http://code.geeksforgeeks.org/oY70kD

// A suffix array based search function to search a given pattern
// 'pat' in given text 'txt' using suffix array suffArr[]
void search(char *pat, char *txt, int *suffArr, int n)
{
    int m = strlen(pat); // get length of pattern, needed for strncmp()

    // Do simple binary search for the pat in txt using the
    // built suffix array
    int l = 0, r = n-1; // Initialize left and right indexes
    while (l <= r)
    {
        // See if 'pat' is prefix of middle suffix in suffix array
        int mid = l + (r - l)/2;
        int res = strncmp(pat, txt+suffArr[mid], m);

        // If match found at the middle, print it and return
        if (res == 0)
        {
            cout << "Pattern found at index " << suffArr[mid];
            return;
        }

        // Move to left half if pattern is alphabetically less than
        // the mid suffix
        if (res < 0) r = mid - 1;

        // Otherwise move to right half
        else l = mid + 1;
    }

    // We reach here if return statement in loop is not executed
    cout << "Pattern not found";
}

// Driver program to test above function
int main()
{
    char txt[] = "banana"; // text
    char pat[] = "nan"; // pattern to be searched in text

    // Build suffix array
    int n = strlen(txt);
    int *suffArr = buildSuffixArray(txt, n);

    // search pat in txt using the built suffix array
    search(pat, txt, suffArr, n);

    return 0;
}
```

[Run on IDE](#)

Output:

Pattern found at index 2

The time complexity of the above search function is $O(m \log n)$. There are more efficient algorithms to search pattern once the suffix array is built. In fact there is a $O(m)$ suffix array based algorithm to search a pattern. We will soon be discussing efficient algorithm for search.

Applications of Suffix Array

Suffix array is an extremely useful data structure, it can be used for a wide range of problems. Following are some

famous problems where Suffix array can be used.

- 1) Pattern Searching
- 2) Finding the longest repeated substring
- 3) Finding the longest common substring
- 4) Finding the longest palindrome in a string

See [this](#) for more problems where Suffix arrays can be used.

This post is a simple introduction. There is a lot to cover in Suffix arrays. We have discussed a $O(n\log n)$ algorithm for Suffix Array construction [here](#). We will soon be discussing more efficient suffix array algorithms.

References:

<http://www.stanford.edu/class/cs97si/suffix-array.pdf>

http://en.wikipedia.org/wiki/Suffix_array

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Ali Tile BY LOLA GROUP

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Advanced Data Structure](#) [Pattern Searching](#)

[Advance Data Structures](#) [Advanced Data Structures](#)

[Pattern Searching](#) [Suffix-Array](#)

Related Posts:

- Second minimum element using minimum comparisons
- Count of distinct substrings of a string using Suffix Trie
- Summed Area Table – Submatrix Summation
- Unrolled Linked List | Set 1 (Introduction)
- proto van Emde Boas Trees | Set 1 (Background and Introduction)
- Implement a Phone Directory
- Binary Indexed Tree : Range Update and Range Queries
- Count and Toggle Queries on a Binary Array

(Login to Rate and Mark)

3.1

Average Difficulty : **3.1/5.0**
Based on **8** vote(s)



Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Anagram Substring Search (Or Search for all permutations)

Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ and its permutations (or anagrams) in $txt[]$. You may assume that $n > m$.

Expected time complexity is $O(n)$

Examples:

- 1) Input: $txt[] = "BACDGABCDA"$ $pat[] = "ABCD"$
 Output: Found at Index 0
 Found at Index 5
 Found at Index 6
- 2) Input: $txt[] = "AAABABAA"$ $pat[] = "AABA"$
 Output: Found at Index 0
 Found at Index 1
 Found at Index 4

We strongly recommend that you click here and practice it, before moving on to the solution.

This problem is slightly different from standard pattern searching problem, here we need to search for anagrams as well. Therefore, we cannot directly apply standard pattern searching algorithms like [KMP](#), [Rabin Karp](#), [Boyer Moore](#), etc.

A simple idea is to modify [Rabin Karp Algorithm](#). For example we can keep the hash value as sum of ASCII values of all characters under modulo of a big prime number. For every character of text, we can add the current character to hash value and subtract the first character of previous window. This solution looks good, but like standard Rabin Karp, the worst case time complexity of this solution is $O(mn)$. The worst case occurs when all hash values match and we one by one match all characters.

We can achieve $O(n)$ time complexity under the assumption that alphabet size is fixed which is typically true as we have maximum 256 possible characters in ASCII. The idea is to use two count arrays:

- 1) The first count array store frequencies of characters in pattern.
- 2) The second count array stores frequencies of characters in current window of text.

The important thing to note is, time complexity to compare two count arrays is O(1) as the number of elements in them are fixed (independent of pattern and text sizes). Following are steps of this algorithm.

1) Store counts of frequencies of pattern in first count array *countP[]*. Also store counts of frequencies of characters in first window of text in array *countTW[]*.

2) Now run a loop from i = M to N-1. Do following in loop.

.....a) If the two count arrays are identical, we found an occurrence.

.....b) Increment count of current character of text in *countTW[]*

.....c) Decrement count of first character in previous window in *countWT[]*

3) The last window is not checked by above loop, so explicitly check it.

Following is C++ implementation of above algorithm.

```
// C++ program to search all anagrams of a pattern in a text
#include<iostream>
#include<cstring>
#define MAX 256
using namespace std;

// This function returns true if contents of arr1[] and arr2[]
// are same, otherwise false.
bool compare(char arr1[], char arr2[])
{
    for (int i=0; i<MAX; i++)
        if (arr1[i] != arr2[i])
            return false;
    return true;
}

// This function search for all permutations of pat[] in txt[]
void search(char *pat, char *txt)
{
    int M = strlen(pat), N = strlen(txt);

    // countP[]: Store count of all characters of pattern
    // countTW[]: Store count of current window of text
    char countP[MAX] = {0}, countTW[MAX] = {0};
    for (int i = 0; i < M; i++)
    {
        (countP[pat[i]])++;
        (countTW[txt[i]])++;
    }

    // Traverse through remaining characters of pattern
    for (int i = M; i < N; i++)
    {
        // Compare counts of current window of text with
        // counts of pattern[]
        if (compare(countP, countTW))
            cout << "Found at Index " << (i - M) << endl;

        // Add current character to current window
        (countTW[txt[i]])++;

        // Remove the first character of previous window
        countTW[txt[i-M]]--;
    }

    // Check for the last window in text
    if (compare(countP, countTW))
        cout << "Found at Index " << (N - M) << endl;
}

/* Driver program to test above function */
int main()
{
    char txt[] = "BACDGABCDA";
    char pat[] = "ABCD";
```

```

search(pat, txt);
return 0;
}

```

[Run on IDE](#)

Output:

```

Found at Index 0
Found at Index 5
Found at Index 6

```

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



GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Pattern Searching](#) [permutation](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

[\(Login to Rate and Mark\)](#)**3.5**Average Difficulty : **3.5/5.0**
Based on **22** vote(s)Add to TODO List
Mark as DONEWriting code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Pattern Searching using a Trie of all Suffixes

Problem Statement: Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. You may assume that $n > m$.

As discussed in the [previous post](#), we discussed that there are two ways efficiently solve the above problem.

1) Preprocess Pattern: [KMP Algorithm](#), [Rabin Karp Algorithm](#), [Finite Automata](#), [Boyer Moore Algorithm](#).

2) Preprocess Text: [Suffix Tree](#)

The best possible time complexity achieved by first (preprocessing pattern) is $O(n)$ and by second (preprocessing text) is $O(m)$ where m and n are lengths of pattern and text respectively.

Note that the second way does the searching only in $O(m)$ time and it is preferred when text doesn't change very frequently and there are many search queries. We have discussed [Suffix Tree \(A compressed Trie of all suffixes of Text\)](#).

Implementation of Suffix Tree may be time consuming for problems to be coded in a technical interview or programming contexts. In this post simple implementation of a [Standard Trie](#) of all Suffixes is discussed. The implementation is close to suffix tree, the only thing is, it's a [simple Trie](#) instead of compressed Trie.

As discussed in [Suffix Tree](#) post, the idea is, every pattern that is present in text (or we can say every substring of text) must be a prefix of one of all possible suffixes. So if we build a Trie of all suffixes, we can find the pattern in $O(m)$ time where m is pattern length.

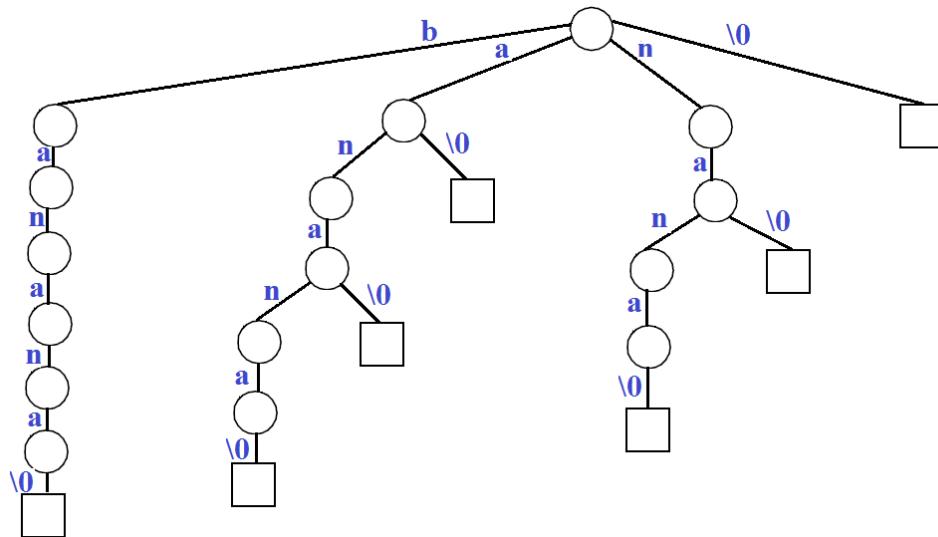
Building a Trie of Suffixes

- 1) Generate all suffixes of given text.
- 2) Consider all suffixes as individual words and build a trie.

Let us consider an example text “banana\0” where ‘\0’ is string termination character. Following are all suffixes of “banana\0”

```
banana\0
anana\0
nana\0
ana\0
na\0
a\0
\0
```

If we consider all of the above suffixes as individual words and build a Trie, we get following.



How to search a pattern in the built Trie?

Following are steps to search a pattern in the built Trie.

- 1) Starting from the first character of the pattern and root of the Trie, do following for every character.
 - a) For the current character of pattern, if there is an edge from the current node, follow the edge.
 - b) If there is no edge, print "pattern doesn't exist in text" and return.
- 2) If all characters of pattern have been processed, i.e., there is a path from root for characters of the given pattern, then print all indexes where pattern is present. To store indexes, we use a list with every node that stores indexes of suffixes starting at the node.

Following is C++ implementation of the above idea.

```
// A simple C++ implementation of substring search using trie of suffixes
#include<iostream>
#include<list>
#define MAX_CHAR 256
using namespace std;

// A Suffix Trie (A Trie of all suffixes) Node
class SuffixTreeNode
{
private:
    SuffixTreeNode *children[MAX_CHAR];
    list<int> *indexes;
public:
    SuffixTreeNode() // Constructor
    {
        // Create an empty linked list for indexes of
        // suffixes starting from this node
        indexes = new list<int>;

        // Initialize all child pointers as NULL
        for (int i = 0; i < MAX_CHAR; i++)
            children[i] = NULL;
    }

    // A recursive function to insert a suffix of the txt
    // in subtree rooted with this node
    void insertSuffix(string suffix, int index);

    // A function to search a pattern in subtree rooted
    // with this node. The function returns pointer to a linked
    // list containing all indexes where pattern is present.
    // The returned indexes are indexes of last characters
    // of matched text.
    list<int>* search(string pat);
};

// A Trie of all suffixes
class SuffixTrie
```

```

{
private:
    SuffixTreeNode root;
public:
    // Constructor (Builds a trie of suffixes of the given text)
    SuffixTrie(string txt)
    {
        // Consider all suffixes of given string and insert
        // them into the Suffix Trie using recursive function
        // insertSuffix() in SuffixTreeNode class
        for (int i = 0; i < txt.length(); i++)
            root.insertSuffix(txt.substr(i), i);
    }

    // Function to searches a pattern in this suffix trie.
    void search(string pat);
};

// A recursive function to insert a suffix of the txt in
// subtree rooted with this node
void SuffixTreeNode::insertSuffix(string s, int index)
{
    // Store index in linked list
    indexes->push_front(index);

    // If string has more characters
    if (s.length() > 0)
    {
        // Find the first character
        char cIndex = s.at(0);

        // If there is no edge for this character, add a new edge
        if (children[cIndex] == NULL)
            children[cIndex] = new SuffixTreeNode();

        // Recur for next suffix
        children[cIndex]->insertSuffix(s.substr(1), index+1);
    }
}

// A recursive function to search a pattern in subtree rooted with
// this node
list<int>* SuffixTreeNode::search(string s)
{
    // If all characters of pattern have been processed,
    if (s.length() == 0)
        return indexes;

    // if there is an edge from the current node of suffix trie,
    // follow the edge.
    if (children[s.at(0)] != NULL)
        return (children[s.at(0)])->search(s.substr(1));

    // If there is no edge, pattern doesn't exist in text
    else return NULL;
}

/* Prints all occurrences of pat in the Suffix Trie S (built for text)*/
void SuffixTrie::search(string pat)
{
    // Let us call recursive search function for root of Trie.
    // We get a list of all indexes (where pat is present in text) in
    // variable 'result'
    list<int> *result = root.search(pat);

    // Check if the list of indexes is empty or not
    if (result == NULL)
        cout << "Pattern not found" << endl;
    else
    {
        list<int>::iterator i;
        int patLen = pat.length();
        for (i = result->begin(); i != result->end(); ++i)
            cout << "Pattern found at position " << *i - patLen << endl;
    }
}

```

```
}

// driver program to test above functions
int main()
{
    // Let us build a suffix trie for text "geeksforgeeks.org"
    string txt = "geeksforgeeks.org";
    SuffixTrie S(txt);

    cout << "Search for 'ee'" << endl;
    S.search("ee");

    cout << "\nSearch for 'geek'" << endl;
    S.search("geek");

    cout << "\nSearch for 'quiz'" << endl;
    S.search("quiz");

    cout << "\nSearch for 'forgeeks'" << endl;
    S.search("forgeeks");

    return 0;
}
```

[Run on IDE](#)

Output:

```
Search for 'ee'
Pattern found at position 9
Pattern found at position 1
```

```
Search for 'geek'
Pattern found at position 8
Pattern found at position 0
```

```
Search for 'quiz'
Pattern not found
```

```
Search for 'forgeeks'
Pattern found at position 5
```

Time Complexity of the above search function is $O(m+k)$ where m is length of the pattern and k is the number of occurrences of pattern in text.

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

Ali Tile BY **LOLA GROUP**

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Advanced Data Structure](#) [Pattern Searching](#) [Pattern Searching](#) [TRIE](#)

Related Posts:

- Second minimum element using minimum comparisons
- Count of distinct substrings of a string using Suffix Trie
- Summed Area Table – Submatrix Summation
- Unrolled Linked List | Set 1 (Introduction)
- proto van Emde Boas Trees | Set 1 (Background and Introduction)
- Implement a Phone Directory
- Binary Indexed Tree : Range Update and Range Queries
- Count and Toggle Queries on a Binary Array

(Login to Rate and Mark)

3.8

Average Difficulty : **3.8/5.0**
Based on **5** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Aho-Corasick Algorithm for Pattern Searching

Given an input text and an array of k words, $\text{arr}[]$, find all occurrences of all words in the input text. Let n be the length of text and m be the total number characters in all words, i.e. $m = \text{length}(\text{arr}[0]) + \text{length}(\text{arr}[1]) + \dots + O(n + \text{length}(\text{arr}[k-1]))$. Here k is total numbers of input words.

Example:

```
Input: text = "ahishers"
      arr[] = {"he", "she", "hers", "his"}
```

Output:

```
Word his appears from 1 to 3
Word he appears from 4 to 5
Word she appears from 3 to 5
Word hers appears from 4 to 7
```

If we use a linear time searching algorithm like **KMP**, then we need to one by one search all words in $\text{text}[]$. This gives us total time complexity as $O(n + \text{length}(\text{word}[0]) + O(n + \text{length}(\text{word}[1]) + O(n + \text{length}(\text{word}[2]) + \dots O(n + \text{length}(\text{word}[k-1]))$. This time complexity can be written as $O(n*k + m)$.

Aho-Corasick Algorithm finds all words in $O(n + m + z)$ time where z is total number of occurrences of words in text. The Aho-Corasick string matching algorithm formed the basis of the original Unix command fgrep.

1. **Preprocessing** : Build an automaton of all words in $\text{arr}[]$ The automaton has mainly three functions:

Go To : This function simply follows edges of Trie of all words in $\text{arr}[]$. It is represented as 2D array $g[][]$ where we store next state for current state and character.

Failure : This function stores all edges that are followed when current character doesn't have edge in Trie. It is represented as 1D array $f[]$ where we store next state for current state.

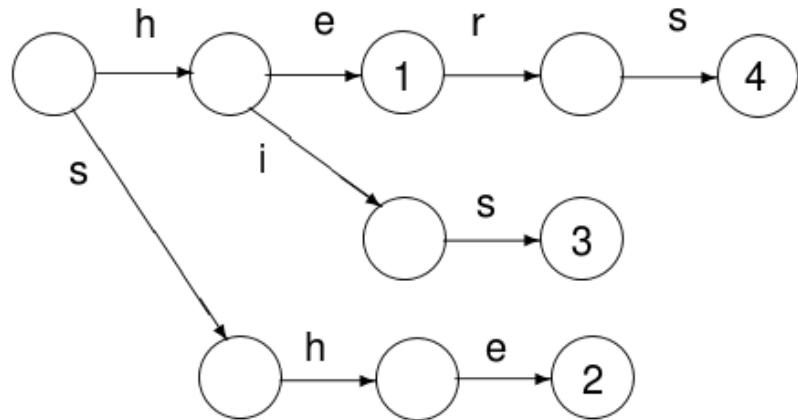
Output : Stores indexes of all words that end at current state. It is represented as 1D array $o[]$ where we store indexes of all matching words as a bitmap for current state.

2. **Matching** : Traverse the given text over built automaton to find all matching words.

Preprocessing:

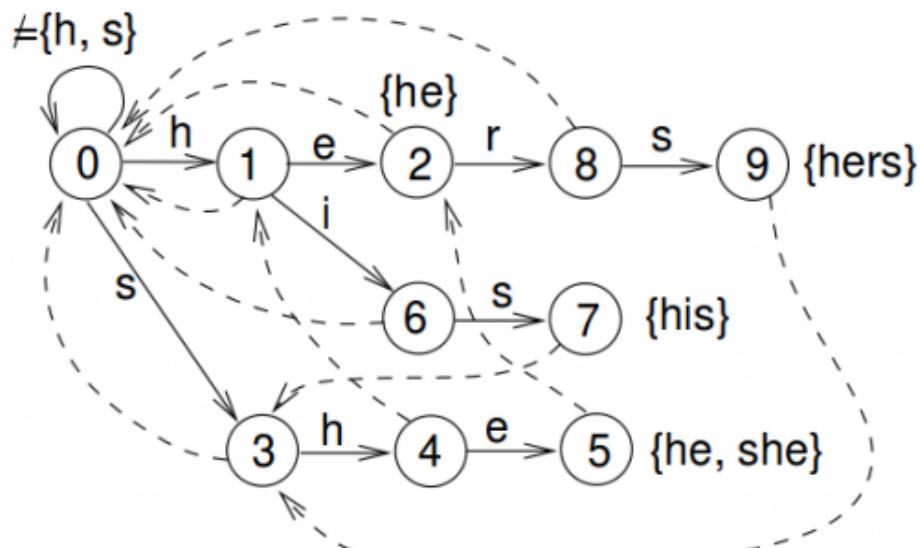
- We first Build a **Trie** (or Keyword Tree) of all words.

Trie for arr[] = {he, she, his, hers}



This part fills entries in goto g[][] and output o[].

- Next we extend Trie into an automaton to support linear time matching.



Dashed arrows are failed transactions.

Normal arrows are Goto (or Trie) transactions

This part fills entries in failure f[] and output o[].

Go to :

We build **Trie**. And for all characters which don't have an edge at root, we add an edge back to root.

Failure :

For a state s, we find the longest proper suffix which is a proper prefix of some pattern. This is done using Breadth First Traversal of Trie.

Output :

For a state s , indexes of all words ending at s are stored. These indexes are stored as bitwise map (by doing bitwise OR of values). This is also computing using Breadth First Traversal with Failure.

Below is C++ implementation of Aho-Corasick Algorithm

```
// C++ program for implementation of Aho Corasick algorithm
// for string matching
using namespace std;
#include <bits/stdc++.h>

// Max number of states in the matching machine.
// Should be equal to the sum of the length of all keywords.
const int MAXS = 500;

// Maximum number of characters in input alphabet
const int MAXC = 26;

// OUTPUT FUNCTION IS IMPLEMENTED USING out[]
// Bit i in this mask is one if the word with index i
// appears when the machine enters this state.
int out[MAXS];

// FAILURE FUNCTION IS IMPLEMENTED USING f[]
int f[MAXS];

// GOTO FUNCTION (OR TRIE) IS IMPLEMENTED USING g[][][]
int g[MAXS][MAXC];

// Builds the string matching machine.
// arr - array of words. The index of each keyword is important:
// "out[state] & (1 << i)" is > 0 if we just found word[i]
// in the text.
// Returns the number of states that the built machine has.
// States are numbered 0 up to the return value - 1, inclusive.
int buildMatchingMachine(string arr[], int k)
{
    // Initialize all values in output function as 0.
    memset(out, 0, sizeof out);

    // Initialize all values in goto function as -1.
    memset(g, -1, sizeof g);

    // Initially, we just have the 0 state
    int states = 1;

    // Construct values for goto function, i.e., fill g[][][]
    // This is same as building a Trie for arr[]
    for (int i = 0; i < k; ++i)
    {
        const string &word = arr[i];
        int currentState = 0;

        // Insert all characters of current word in arr[]
        for (int j = 0; j < word.size(); ++j)
        {
            int ch = word[j] - 'a';

            // Allocate a new node (create a new state) if a
            // node for ch doesn't exist.
            if (g[currentState][ch] == -1)
                g[currentState][ch] = states++;

            currentState = g[currentState][ch];
        }

        // Add current word in output function
        out[currentState] |= (1 << i);
    }

    // For all characters which don't have an edge from
    // root (or state 0) in Trie, add a goto edge to state
}
```

```

// 0 itself
for (int ch = 0; ch < MAXC; ++ch)
    if (g[0][ch] == -1)
        g[0][ch] = 0;

// Now, let's build the failure function

// Initialize values in fail function
memset(f, -1, sizeof f);

// Failure function is computed in breadth first order
// using a queue
queue<int> q;

// Iterate over every possible input
for (int ch = 0; ch < MAXC; ++ch)
{
    // All nodes of depth 1 have failure function value
    // as 0. For example, in above diagram we move to 0
    // from states 1 and 3.
    if (g[0][ch] != 0)
    {
        f[g[0][ch]] = 0;
        q.push(g[0][ch]);
    }
}

// Now queue has states 1 and 3
while (q.size())
{
    // Remove the front state from queue
    int state = q.front();
    q.pop();

    // For the removed state, find failure function for
    // all those characters for which goto function is
    // not defined.
    for (int ch = 0; ch <= MAXC; ++ch)
    {
        // If goto function is defined for character 'ch'
        // and 'state'
        if (g[state][ch] != -1)
        {
            // Find failure state of removed state
            int failure = f[state];

            // Find the deepest node labeled by proper
            // suffix of string from root to current
            // state.
            while (g[failure][ch] == -1)
                failure = f[failure];

            failure = g[failure][ch];
            f[g[state][ch]] = failure;

            // Merge output values
            out[g[state][ch]] |= out[failure];

            // Insert the next level node (of Trie) in Queue
            q.push(g[state][ch]);
        }
    }
}

return states;
}

// Returns the next state the machine will transition to using goto
// and failure functions.
// currentState - The current state of the machine. Must be between
//                 0 and the number of states - 1, inclusive.
// nextInput - The next character that enters into the machine.
int findNextState(int currentState, char nextInput)
{

```

```

int answer = currentState;
int ch = nextInput - 'a';

// If goto is not defined, use failure function
while (g[answer][ch] == -1)
    answer = f[answer];

return g[answer][ch];
}

// This function finds all occurrences of all array words
// in text.
void searchWords(string arr[], int k, string text)
{
    // Preprocess patterns.
    // Build machine with goto, failure and output functions
    buildMatchingMachine(arr, k);

    // Initialize current state
    int currentState = 0;

    // Traverse the text through the nult machine to find
    // all occurrences of words in arr[]
    for (int i = 0; i < text.size(); ++i)
    {
        currentState = findNextState(currentState, text[i]);

        // If match not found, move to next state
        if (out[currentState] == 0)
            continue;

        // Match found, print all matching words of arr[]
        // using output function.
        for (int j = 0; j < k; ++j)
        {
            if (out[currentState] & (1 << j))
            {
                cout << "Word " << arr[j] << " appears from "
                    << i - arr[j].size() + 1 << " to " << i << endl;
            }
        }
    }
}

// Driver program to test above
int main()
{
    string arr[] = {"he", "she", "hers", "his"};
    string text = "ahishers";
    int k = sizeof(arr)/sizeof(arr[0]);

    searchWords(arr, k, text);

    return 0;
}

```

Run on IDE

Output:

```

Word his appears from 1 to 3
Word he appears from 4 to 5
Word she appears from 3 to 5
Word hers appears from 4 to 7

```

Source:

<http://www.cs.uku.fi/~kilpelai/BSA05/lectures/slides04.pdf>

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



GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- [kasai's Algorithm for Construction of LCP array from Suffix Array](#)
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 3

(Login to Rate and Mark)

4.6

Average Difficulty : 4.6/5.0
Based on 15 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

kasai's Algorithm for Construction of LCP array from Suffix Array

Background

Suffix Array : A suffix array is a sorted array of all suffixes of a given string.

Let the given string be "banana".

0 banana	Sort the Suffixes	5 a
1 anana		3 ana
2 nana	----->	1 anana
3 ana	alphabetically	0 banana
4 na		4 na
5 a		2 nana

The suffix array for "banana" :

suffix[] = {5, 3, 1, 0, 4, 2}

We have discussed [Suffix Array](#) and its O(nLogn) construction .

Once Suffix array is built, we can use it to efficiently search a pattern in a text. For example, we can use Binary Search to find a pattern (Complete code for the same is discussed [here](#))

LCP Array

The Binary Search based solution discussed [here](#) takes O(m*Logn) time where m is length of the pattern to be searched and n is length of the text. With the help of LCP array, we can search a pattern in O(m + Log n) time. For example, if our task is to search "ana" in "banana", m = 3, n = 5.

LCP Array is an array of size n (like Suffix Array). A value lcp[i] indicates length of the longest common prefix of the suffixes indexed by suffix[i] and suffix[i+1]. suffix[n-1] is not defined as there is no suffix after it.

```
txt[0..n-1] = "banana"
suffix[] = {5, 3, 1, 0, 4, 2}
lcp[] = {1, 3, 0, 0, 2, 0}
```

Suffixes represented by suffix array in order are:

{"a", "ana", "anana", "banana", "na", "nana"}

```
lcp[0] = Longest Common Prefix of "a" and "ana"      = 1
lcp[1] = Longest Common Prefix of "ana" and "anana" = 3
lcp[2] = Longest Common Prefix of "anana" and "banana" = 0
```

```

lcp[3] = Longest Common Prefix of "banana" and "na" = 0
lcp[4] = Longest Common Prefix of "na" and "nana" = 2
lcp[5] = Longest Common Prefix of "nana" and None = 0

```

How to construct LCP array?

LCP array construction is done two ways:

- 1) Compute the LCP array as a byproduct to the suffix array (Manber & Myers Algorithm)
- 2) Use an already constructed suffix array in order to compute the LCP values. (Kasai Algorithm).

There exist algorithms that can construct Suffix Array in $O(n)$ time and therefore we can always construct LCP array in $O(n \log n)$ time. But in the below implementation, a $O(n \log n)$ algorithm is discussed.

kasai's Algorithm

In this article kasai's Algorithm is discussed. The algorithm constructs LCP array from suffix array and input text in $O(n)$ time. The idea is based on below fact:

Let lcp of suffix beginning at $txt[i]$ be k . If k is greater than 0, then lcp for suffix beginning at $txt[i+1]$ will be at-least $k-1$. The reason is, relative order of characters remain same. If we delete the first character from both suffixes, we know that at least k characters will match. For example for substring "ana", lcp is 3, so for string "na" lcp will be at-least 2. Refer [this](#) for proof.

Below is C++ implementation of Kasai's algorithm.

```

// C++ program for building LCP array for given text
#include <bits/stdc++.h>
using namespace std;

// Structure to store information of a suffix
struct suffix
{
    int index; // To store original index
    int rank[2]; // To store ranks and next rank pair
};

// A comparison function used by sort() to compare two suffixes
// Compares two pairs, returns 1 if first pair is smaller
int cmp(struct suffix a, struct suffix b)
{
    return (a.rank[0] == b.rank[0])? (a.rank[1] < b.rank[1] ?1: 0):
        (a.rank[0] < b.rank[0] ?1: 0);
}

// This is the main function that takes a string 'txt' of size n as an
// argument, builds and return the suffix array for the given string
vector<int> buildSuffixArray(string txt, int n)
{
    // A structure to store suffixes and their indexes
    struct suffix suffixes[n];

    // Store suffixes and their indexes in an array of structures.
    // The structure is needed to sort the suffixes alphabetically
    // and maintain their old indexes while sorting
    for (int i = 0; i < n; i++)
    {
        suffixes[i].index = i;
        suffixes[i].rank[0] = txt[i] - 'a';
        suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1] - 'a'): -1;
    }

    // Sort the suffixes using the comparison function
    // defined above.
    sort(suffixes, suffixes+n, cmp);

    // At this point, all suffixes are sorted according to first
    // 2 characters. Let us sort suffixes according to first 4
}

```

```

// characters, then first 8 and so on
int ind[n]; // This array is needed to get the index in suffixes[]
// from original index. This mapping is needed to get
// next suffix.
for (int k = 4; k < 2*n; k = k*2)
{
    // Assigning rank and index values to first suffix
    int rank = 0;
    int prev_rank = suffixes[0].rank[0];
    suffixes[0].rank[0] = rank;
    ind[suffixes[0].index] = 0;

    // Assigning rank to suffixes
    for (int i = 1; i < n; i++)
    {
        // If first rank and next ranks are same as that of previous
        // suffix in array, assign the same new rank to this suffix
        if (suffixes[i].rank[0] == prev_rank &&
            suffixes[i].rank[1] == suffixes[i-1].rank[1])
        {
            prev_rank = suffixes[i].rank[0];
            suffixes[i].rank[0] = rank;
        }
        else // Otherwise increment rank and assign
        {
            prev_rank = suffixes[i].rank[0];
            suffixes[i].rank[0] = ++rank;
        }
        ind[suffixes[i].index] = i;
    }

    // Assign next rank to every suffix
    for (int i = 0; i < n; i++)
    {
        int nextindex = suffixes[i].index + k/2;
        suffixes[i].rank[1] = (nextindex < n)?
            suffixes[ind[nextindex]].rank[0]: -1;
    }

    // Sort the suffixes according to first k characters
    sort(suffixes, suffixes+n, cmp);
}

// Store indexes of all sorted suffixes in the suffix array
vector<int>suffixArr;
for (int i = 0; i < n; i++)
    suffixArr.push_back(suffixes[i].index);

// Return the suffix array
return suffixArr;
}

/* To construct and return LCP */
vector<int> kasai(string txt, vector<int> suffixArr)
{
    int n = suffixArr.size();

    // To store LCP array
    vector<int> lcp(n, 0);

    // An auxiliary array to store inverse of suffix array
    // elements. For example if suffixArr[0] is 5, the
    // invSuff[5] would store 0. This is used to get next
    // suffix string from suffix array.
    vector<int> invSuff(n, 0);

    // Fill values in invSuff[]
    for (int i=0; i < n; i++)
        invSuff[suffixArr[i]] = i;

    // Initialize length of previous LCP
    int k = 0;

    // Process all suffixes one by one starting from

```

```

// first suffix in txt[]
for (int i=0; i<n; i++)
{
    /* If the current suffix is at n-1, then we don't
       have next substring to consider. So lcp is not
       defined for this substring, we put zero. */
    if (invSuff[i] == n-1)
    {
        k = 0;
        continue;
    }

    /* j contains index of the next substring to
       be considered to compare with the present
       substring, i.e., next string in suffix array */
    int j = suffixArr[invSuff[i]+1];

    // Directly start matching from k'th index as
    // at-least k-1 characters will match
    while (i+k<n && j+k<n && txt[i+k]==txt[j+k])
        k++;

    lcp[invSuff[i]] = k; // lcp for the present suffix.

    // Deleting the starting character from the string.
    if (k>0)
        k--;
}

// return the constructed lcp array
return lcp;
}

// Utility function to print an array
void printArr(vector<int>arr, int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

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

    vector<int>suffixArr = buildSuffixArray(str, str.length());
    int n = suffixArr.size();

    cout << "Suffix Array : \n";
    printArr(suffixArr, n);

    vector<int>lcp = kasai(str, suffixArr);

    cout << "\nLCP Array : \n";
    printArr(lcp, n);
    return 0;
}

```

Run on IDE

Output:

Suffix Array :

5 3 1 0 4 2

LCP Array :

1 3 0 0 2 0

Illustration:

```
txt[] = "banana", suffix[] = {5, 3, 1, 0, 4, 2|
```

Suffix array represents
 {"a", "ana", "anana", "banana", "na", "nana"}

Inverse Suffix Array would be
 invSuff[] = {3, 2, 5, 1, 4, 0}

LCP values are evaluated in below order

We first compute LCP of first suffix in text which is “**banana**”. We need next suffix in suffix array to compute LCP (Remember lcp[i] is defined as Longest Common Prefix of suffix[i] and suffix[i+1]). **To find the next suffix in suffixArr[], we use SuffInv[]**. The next suffix is “na”. Since there is no common prefix between “banana” and “na”, the value of LCP for “banana” is 0 and it is at index 3 in suffix array, so we fill **lcp[3]** as 0.

Next we compute LCP of second suffix which “**anana**”. Next suffix of “anana” in suffix array is “banana”. Since there is no common prefix, the value of LCP for “anana” is 0 and it is at index 2 in suffix array, so we fill **lcp[2]** as 0.

Next we compute LCP of third suffix which “**nana**”. Since there is no next suffix, the value of LCP for “nana” is not defined. We fill **lcp[5]** as 0.

Next suffix in text is “ana”. Next suffix of “**ana**” in suffix array is “anana”. Since there is a common prefix of length 3, the value of LCP for “ana” is 3. We fill **lcp[1]** as 3.

Now we lcp for next suffix in text which is “**na**”. This is where Kasai’s algorithm uses the trick that LCP value must be at least 2 because previous LCP value was 3. Since there is no character after “na”, final value of LCP is 2. We fill **lcp[4]** as 2.

Next suffix in text is “**a**”. LCP value must be at least 1 because previous value was 2. Since there is no character after “a”, final value of LCP is 1. We fill **lcp[0]** as 1.

We will soon be discussing implementation of search with the help of LCP array and how LCP array helps in reducing time complexity to O(m + Log n).

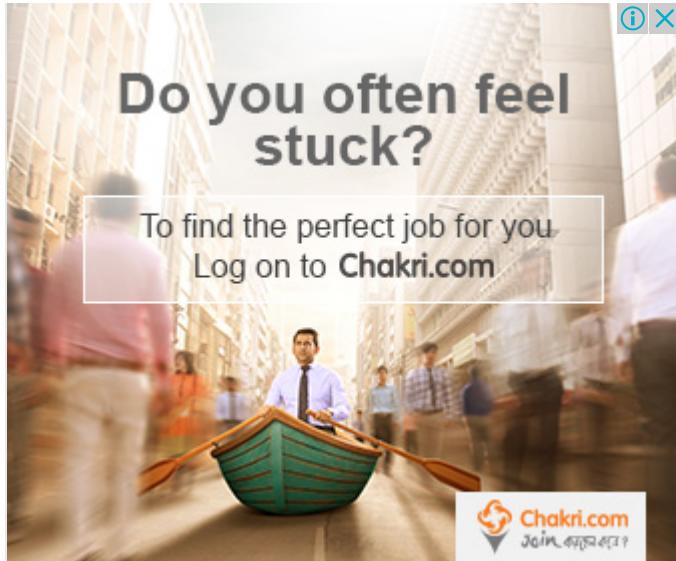
References:

<http://web.stanford.edu/class/cs97si/suffix-array.pdf>

<http://www.mi.fu-berlin.de/wiki/pub/ABI/RnaSeqP4/suffix-array.pdf>

<http://codeforces.com/blog/entry/12796>

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



GATE CS Corner Company Wise Coding Practice

[Advanced Data Structure](#) [Pattern Searching](#) [Technical Scripter](#)

Related Posts:

- Second minimum element using minimum comparisons
- Count of distinct substrings of a string using Suffix Trie
- Summed Area Table – Submatrix Summation
- Unrolled Linked List | Set 1 (Introduction)
- proto van Emde Boas Trees | Set 1 (Background and Introduction)
- Implement a Phone Directory
- Binary Indexed Tree : Range Update and Range Queries
- Count and Toggle Queries on a Binary Array

(Login to Rate and Mark)

3.4

Average Difficulty : **3.4/5.0**
Based on **5** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Z algorithm (Linear time pattern searching Algorithm)

This algorithm finds all occurrences of a pattern in a text in linear time. Let length of text be n and of pattern be m , then total time taken is $O(m + n)$ with linear space complexity. Now we can see that both time and space complexity is same as KMP algorithm but this algorithm is Simpler to understand.

In this algorithm, we construct a Z array.

What is Z Array?

For a string $\text{str}[0..n-1]$, Z array is of same length as string. An element $Z[i]$ of Z array stores length of the longest substring starting from $\text{str}[i]$ which is also a prefix of $\text{str}[0..n-1]$. The first entry of Z array is meaning less as complete string is always prefix of itself.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text	a	a	b	c	a	a	b	x	a	a	a	z
Z values	X	1	0	0	3	1	0	0	2	2	1	0

More Examples:

```
str = "aaaaaa"
```

```
Z[] = {x, 5, 4, 3, 2, 1}
```

```
str = "aabbaacd"
```

```
Z[] = {x, 1, 0, 2, 1, 0, 0}
```

```
str = "abababab"
```

```
Z[] = {x, 0, 6, 0, 4, 0, 2, 0}
```

How is Z array helpful in Searching Pattern in Linear time?

The idea is to concatenate pattern and text, and create a string “P\$T” where P is pattern, \$ is a special character should not be present in pattern and text, and T is text. Build the Z array for concatenated string. In Z array, if Z value at any point is equal to pattern length, then pattern is present at that point.

Example:

```
Pattern P = "aab", Text T = "baabaa"
```

The concatenated string is = "aab\$baabaa"

```
Z array for above concatenated string is {x, 1, 0, 0, 0,  
3, 1, 0, 2, 1}.
```

Since length of pattern is 3, the value 3 in Z array indicates presence of pattern.

How to construct Z array?

A Simple Solution is two run two nested loops, the outer loop goes to every index and the inner loop finds length of the longest prefix that matches substring starting at current index. The time complexity of this solution is $O(n^2)$.

We can construct Z array in linear time.

The idea is to maintain an interval $[L, R]$ which is the interval with max R such that $[L, R]$ is prefix substring (substring which is also prefix).

Steps for maintaining this interval are as follows -

- 1) If $i > R$ then there is no prefix substring that starts before i and ends after i , so we reset L and R and compute new $[L, R]$ by comparing $\text{str}[0..]$ to $\text{str}[i..]$ and get $Z[i]$ ($= R-L+1$).
- 2) If $i \leq R$ then let $K = i-L$, now $Z[i] \geq \min(Z[K], R-i+1)$ because $\text{str}[i..]$ matches with $\text{str}[K..]$ for atleast $R-i+1$ characters (they are in $[L, R]$ interval which we know is a prefix substring).
Now two sub cases arise -
 - a) If $Z[K] < R-i+1$ then there is no prefix substring starting at $\text{str}[i]$ (otherwise $Z[K]$ would be larger) so $Z[i] = Z[K]$ and interval $[L, R]$ remains same.
 - b) If $Z[K] \geq R-i+1$ then it is possible to extend the $[L, R]$ interval thus we will set L as i and start matching from $\text{str}[R]$ onwards and get new R then we will update interval $[L, R]$ and calculate $Z[i]$ ($= R-L+1$).

For better understanding of above step by step procedure please check this animation – <http://www.utdallas.edu/~besp/demo/John2010/z-algorithm.htm>

The algorithm runs in linear time because we never compare character less than R and with matching we increase R by one so there are at most T comparisons. In mismatch case, mismatch happen only once for each i (because of which R stops), that's another at most T comparison making overall linear complexity.

Below is C++ implementation of Z algorithm for pattern searching.

```
// A C++ program that implements Z algorithm for pattern searching
#include<iostream>
using namespace std;

void getZarr(string str, int Z[]);
// prints all occurrences of pattern in text using Z algo
void search(string text, string pattern)
{
    // Create concatenated string "P$T"
    string concat = pattern + "$" + text;
    int l = concat.length();

    // Construct Z array
    int Z[l];
    getZarr(concat, Z);

    // now looping through Z array for matching condition
    for (int i = 0; i < l; ++i)
    {
        // if Z[i] (matched region) is equal to pattern
        // length we got the pattern
    }
}
```

```

        if (Z[i] == pattern.length())
            cout << "Pattern found at index "
                << i - pattern.length() -1 << endl;
    }

// Fills Z array for given string str[]
void getZarr(string str, int Z[])
{
    int n = str.length();
    int L, R, k;

    // [L,R] make a window which matches with prefix of s
    L = R = 0;
    for (int i = 1; i < n; ++i)
    {
        // if i>R nothing matches so we will calculate.
        // Z[i] using naive way.
        if (i > R)
        {
            L = R = i;

            // R-L = 0 in starting, so it will start
            // checking from 0'th index. For example,
            // for "ababab" and i = 1, the value of R
            // remains 0 and Z[i] becomes 0. For string
            // "aaaaaa" and i = 1, Z[i] and R become 5
            while (R < n && str[R-L] == str[R])
                R++;
            Z[i] = R-L;
            R--;
        }
        else
        {
            // k = i-L so k corresponds to number which
            // matches in [L,R] interval.
            k = i-L;

            // if Z[k] is less than remaining interval
            // then Z[i] will be equal to Z[k].
            // For example, str = "ababab", i = 3, R = 5
            // and L = 2
            if (Z[k] < R-i+1)
                Z[i] = Z[k];

            // For example str = "aaaaaa" and i = 2, R is 5,
            // L is 0
            else
            {
                // else start from R and check manually
                L = i;
                while (R < n && str[R-L] == str[R])
                    R++;
                Z[i] = R-L;
                R--;
            }
        }
    }
}

// Driver program
int main()
{
    string text = "GEEKS FOR GEEKS";
    string pattern = "GEEK";
    search(text, pattern);
    return 0;
}

```

[Run on IDE](#)

Output:

Pattern found at index 0
 Pattern found at index 10

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



China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Pattern Searching](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 3

(Login to Rate and Mark)

3.5 Average Difficulty : 3.5/5.0
 Based on 10 vote(s)

 Add to TODO List
 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 1

Given a string, find the longest substring which is palindrome.

- if the given string is “forgeeksskeegfor”, the output should be “geeksskeeg”
- if the given string is “abaaba”, the output should be “abaaba”
- if the given string is “abababa”, the output should be “abababa”
- if the given string is “abcbabcbabcba”, the output should be “abcbabcbabcba”

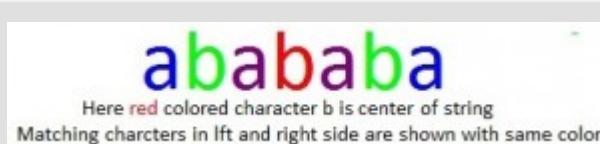
We have already discussed Naïve [$O(n^3)$] and quadratic [$O(n^2)$] approaches at [Set 1](#) and [Set 2](#).

In this article, we will talk about [Manacher's algorithm](#) which finds Longest Palindromic Substring in linear time.

One way ([Set 2](#)) to find a palindrome is to start from the center of the string and compare characters in both directions one by one. If corresponding characters on both sides (left and right of the center) match, then they will make a palindrome.

Let's consider string “abababa”.

Here center of the string is 4th character (with index 3) b. If we match characters in left and right of the center, all characters match and so string “abababa” is a palindrome.



Here center position is not only the actual string character position but it could be the position between two characters also.

Consider string “abaaba” of even length. This string is palindrome around the position between 3rd and 4th characters a and a respectively.



To find Longest Palindromic Substring of a string of length N, one way is take each possible $2*N + 1$ centers (the N character positions, N-1 between two character positions and 2 positions at left and right ends), do the character match in both left and right directions at each $2*N+ 1$ centers and keep track of LPS. This approach takes $O(N^2)$ time and that's what we are doing in [Set 2](#).

Let's consider two strings "abababa" and "abaaba" as shown below:



In these two strings, left and right side of the center positions (position 7 in 1st string and position 6 in 2nd string) are symmetric. Why? Because the whole string is palindrome around the center position.

If we need to calculate Longest Palindromic Substring at each $2*N+1$ positions from left to right, then palindrome's symmetric property could help to avoid some of the unnecessary computations (i.e. character comparison). If there is a palindrome of some length L centered at any position P, then we may not need to compare all characters in left and right side at position P+1. We already calculated LPS at positions before P and they can help to avoid some of the comparisons after position P.

This use of information from previous positions at a later point of time makes the Manacher's algorithm linear. In **Set 2**, there is no reuse of previous information and so that is quadratic.

Manacher's algorithm is probably considered complex to understand, so here we will discuss it in as detailed way as we can. Some of its portions may require multiple reading to understand it properly.

Let's look at string "abababa". In 3rd figure above, 15 center positions are shown. We need to calculate length of longest palindromic string at each of these positions.

- At position 0, there is no LPS at all (no character on left side to compare), so length of LPS will be 0.
- At position 1, LPS is a, so length of LPS will be 1.
- At position 2, there is no LPS at all (left and right characters a and b don't match), so length of LPS will be 0.
- At position 3, LPS is aba, so length of LPS will be 3.
- At position 4, there is no LPS at all (left and right characters b and a don't match), so length of LPS will be 0.
- At position 5, LPS is ababa, so length of LPS will be 5.

..... and so on

We store all these palindromic lengths in an array, say L. Then string S and LPS Length L look like below:

String S		a		b		a		b		a		b		a	
LPS Length L	0	1	0	3	0	5	0	7	0	5	0	3	0	1	0
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Similarly, LPS Length L of string "abaaba" will look like:

String S		a		b		a		a		b		a	
LPS Length L	0	1	0	3	0	1	6	1	0	3	0	1	0
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12

In LPS Array L:

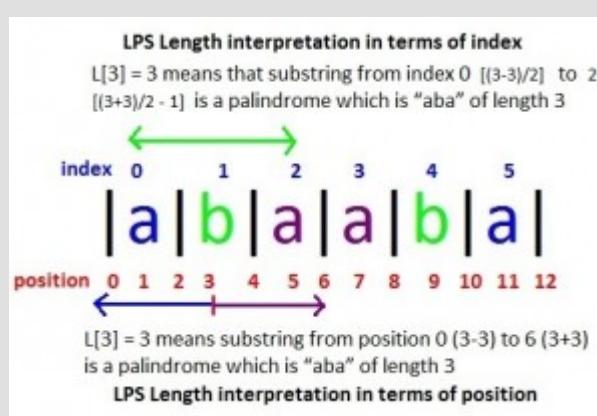
- LPS length value at odd positions (the actual character positions) will be odd and greater than or equal to 1 (1 will come from the center character itself if nothing else matches in left and right side of it)
- LPS length value at even positions (the positions between two characters, extreme left and right positions) will be even and greater than or equal to 0 (0 will come when there is no match in left and right side)

Position and index for the string are two different things here. For a given string S of length N, indexes will be from 0 to N-1 (total N indexes) and positions will be from 0 to 2*N (total 2*N+1 positions).

LPS length value can be interpreted in two ways, one in terms of index and second in terms of position. LPS value d at position i ($L[i] = d$) tells that:

- Substring from position i-d to i+d is a palindrome of length d (in terms of position)
- Substring from index $(i-d)/2$ to $[(i+d)/2 - 1]$ is a palindrome of length d (in terms of index)

e.g. in string "abaaba", $L[3] = 3$ means substring from position 0 (3-3) to 6 (3+3) is a palindrome which is "aba" of length 3, it also means that substring from index 0 $[(3-3)/2]$ to 2 $[(3+3)/2 - 1]$ is a palindrome which is "aba" of length 3.



Now the main task is to compute LPS array efficiently. Once this array is computed, LPS of string S will be centered at position with maximum LPS length value.

We will see it in [Part 2](#).

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



GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Strings](#) [palindrome](#)

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- Kasai's Algorithm for Construction of LCP array from Suffix Array
- Search a Word in a 2D Grid of characters
- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

3.6

Average Difficulty : **3.6/5.0**
Based on **16** vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

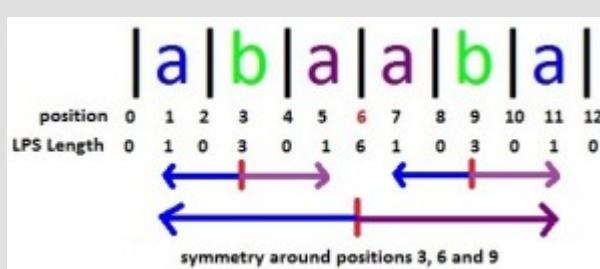
Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 2

In [Manacher's Algorithm – Part 1](#), we gone through some of the basics and LPS length array.

Here we will see how to calculate LPS length array efficiently.

To calculate LPS array efficiently, we need to understand how LPS length for any position may relate to LPS length value of any previous already calculated position.

For string “abaaba”, we see following:



If we look around position 3:

- LPS length value at position 2 and position 4 are same
- LPS length value at position 1 and position 5 are same

We calculate LPS length values from left to right starting from position 0, so we can see if we already know LPS length values at positions 1, 2 and 3 already then we may not need to calculate LPS length at positions 4 and 5 because they are equal to LPS length values at corresponding positions on left side of position 3.

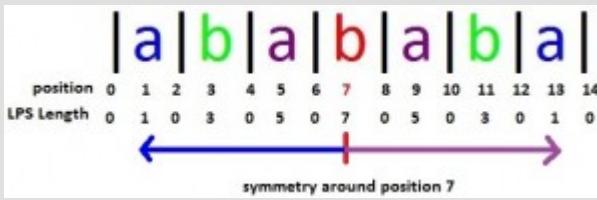
If we look around position 6:

- LPS length value at position 5 and position 7 are same
- LPS length value at position 4 and position 8 are same

..... and so on.

If we already know LPS length values at positions 1, 2, 3, 4, 5 and 6 already then we may not need to calculate LPS length at positions 7, 8, 9, 10 and 11 because they are equal to LPS length values at corresponding positions on left side of position 6.

For string “abababa”, we see following:



If we already know LPS length values at positions 1, 2, 3, 4, 5, 6 and 7 already then we may not need to calculate LPS length at positions 8, 9, 10, 11, 12 and 13 because they are equal to LPS length values at corresponding positions on left side of position 7.

Can you see why LPS length values are symmetric around positions 3, 6, 9 in string "abaaba"? That's because there is a palindromic substring around these positions. Same is the case in string "abababa" around position 7.

Is it always true that LPS length values around at palindromic center position are always symmetric (same)?

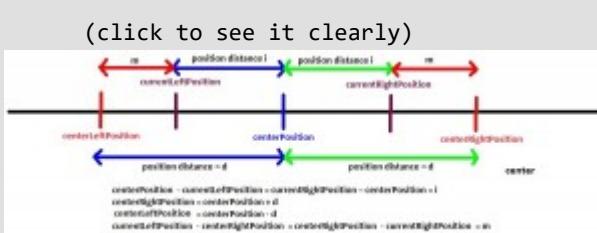
Answer is NO.

Look at positions 3 and 11 in string "abababa". Both positions have LPS length 3. Immediate left and right positions are symmetric (with value 0), but not the next one. Positions 1 and 5 (around position 3) are not symmetric. Similarly, positions 9 and 13 (around position 11) are not symmetric.

At this point, we can see that if there is a palindrome in a string centered at some position, then LPS length values around the center position may or may not be symmetric depending on some situation. If we can identify the situation when left and right positions WILL BE SYMMETRIC around the center position, we NEED NOT calculate LPS length of the right position because it will be exactly same as LPS value of corresponding position on the left side which is already known. And this fact where we are avoiding LPS length computation at few positions makes Manacher's Algorithm linear.

In situations when left and right positions WILL NOT BE SYMMETRIC around the center position, we compare characters in left and right side to find palindrome, but here also algorithm tries to avoid certain no of comparisons. We will see all these scenarios soon.

Let's introduce few terms to proceed further:



- **centerPosition** – This is the position for which LPS length is calculated and let's say LPS length at centerPosition is d (i.e. $L[centerPosition] = d$)
- **centerRightPosition** – This is the position which is right to the centerPosition and d position away from centerPosition (i.e. $centerRightPosition = centerPosition + d$)
- **centerLeftPosition** – This is the position which is left to the centerPosition and d position away from centerPosition (i.e. $centerLeftPosition = centerPosition - d$)
- **currentRightPosition** – This is the position which is right of the centerPosition for which LPS length is not yet known and has to be calculated
- **currentLeftPosition** – This is the position on the left side of centerPosition which corresponds to the currentRightPosition

- centerPosition – currentLeftPosition = currentRightPosition – centerPosition**
- currentLeftPosition = 2* centerPosition – currentRightPosition**
- **i-left palindrome** – The palindrome i positions left of centerPosition, i.e. at currentLeftPosition
- **i-right palindrome** – The palindrome i positions right of centerPosition, i.e. at currentRightPosition
- **center palindrome** – The palindrome at centerPosition

When we are at centerPosition for which LPS length is known, then we also know LPS length of all positions smaller than centerPosition. Let's say LPS length at centerPosition is d, i.e.

$$L[\text{centerPosition}] = d$$

It means that substring between positions "centerPosition-d" to "centerPosition+d" is a palindrom.

Now we proceed further to calculate LPS length of positions greater than centerPosition.

Let's say we are at currentRightPosition ($>$ centerPosition) where we need to find LPS length.

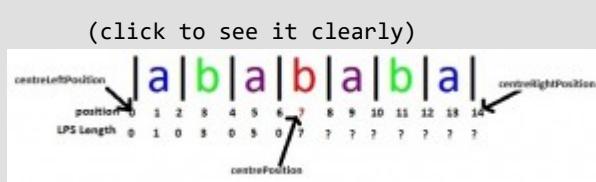
For this we look at LPS length of currentLeftPosition which is already calculated.

If LPS length of currentLeftPosition is less than "centerRightPosition – currentRightPosition", then LPS length of currentRightPosition will be equal to LPS length of currentLeftPosition. So

$$L[\text{currentRightPosition}] = L[\text{currentLeftPosition}] \quad \text{if} \quad L[\text{currentLeftPosition}] < \text{centerRightPosition} - \text{currentRightPosition}$$

This is **Case 1**.

Let's consider below scenario for string "abababa":



We have calculated LPS length up-to position 7 where $L[7] = 7$, if we consider position 7 as centerPosition, then centerLeftPosition will be 0 and centerRightPosition will be 14.

Now we need to calculate LPS length of other positions on the right of centerPosition.

For $\text{currentRightPosition} = 8$, $\text{currentLeftPosition}$ is 6 and $L[\text{currentLeftPosition}] = 0$

Also $\text{centerRightPosition} - \text{currentRightPosition} = 14 - 8 = 6$

Case 1 applies here and so $L[\text{currentRightPosition}] = L[8] = 0$

Case 1 applies to positions 10 and 12, so,

$$L[10] = L[4] = 0$$

$$L[12] = L[2] = 0$$

If we look at position 9, then:

$$\text{currentRightPosition} = 9$$

$$\text{currentLeftPosition} = 2 * \text{centerPosition} - \text{currentRightPosition} = 2 * 7 - 9 = 5$$

$$\text{centerRightPosition} - \text{currentRightPosition} = 14 - 9 = 5$$

Here $L[\text{currentLeftPosition}] = \text{centerRightPosition} - \text{currentRightPosition}$, so Case 1 doesn't apply here. Also note that centerRightPosition is the extreme end position of the string. That means center palindrome is suffix of input string. In that case, $L[\text{currentRightPosition}] = L[\text{currentLeftPosition}]$. This is **Case 2**.

Case 2 applies to positions 9, 11, 13 and 14, so:

$$L[9] = L[5] = 5$$

$$L[11] = L[3] = 3$$

$L[13] = L[1] = 1$

$L[14] = L[0] = 0$

What is really happening in Case 1 and Case 2? This is just utilizing the palindromic symmetric property and without any character match, it is finding LPS length of new positions.

When a bigger length palindrome contains a smaller length palindrome centered at left side of its own center, then based on symmetric property, there will be another same smaller palindrome centered on the right of bigger palindrome center. If left side smaller palindrome is not prefix of bigger palindrome, then **Case 1** applies and if it is a prefix AND bigger palindrome is suffix of the input string itself, then **Case 2** applies.

*The longest palindrome i places to the right of the current center (the i -right palindrome) is as long as the longest palindrome i places to the left of the current center (the i -left palindrome) if the i -left palindrome is completely contained in the longest palindrome around the current center (the center palindrome) and the i -left palindrome is not a prefix of the center palindrome (**Case 1**) or (i.e. when i -left palindrome is a prefix of center palindrome) if the center palindrome is a suffix of the entire string (**Case 2**).*

In Case 1 and Case 2, i -right palindrome can't expand more than corresponding i -left palindrome (can you visualize why it can't expand more?), and so LPS length of i -right palindrome is exactly same as LPS length of i -left palindrome.

Here both i -left and i -right palindromes are completely contained in center palindrome (i.e. $L[\text{currentLeftPosition}] \leq \text{centerRightPosition} - \text{currentRightPosition}$)

Now if i -left palindrome is not a prefix of center palindrome ($L[\text{currentLeftPosition}] < \text{centerRightPosition} - \text{currentRightPosition}$), that means that i -left palindrome was not able to expand up-to position $\text{centerLeftPosition}$.

If we look at following with $\text{centerPosition} = 11$, then

		(click to see it clearly)																											
String S		c d b s b c b s b d b s b																											
LPS Length L	0	1	0	1	0	1	0	3	0	1	0	9	0	1	0	3	0	1	0	7	0	1	0	3	0	1	0		
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26		

$\text{centerLeftPosition}$ would be $11 - 9 = 2$, and $\text{centerRightPosition}$ would be $11 + 9 = 20$

If we take $\text{currentRightPosition} = 15$, it's $\text{currentLeftPosition}$ is 7. Case 1 applies here and so $L[15] = 3$. i -left palindrome at position 7 is "bab" which is completely contained in center palindrome at position 11 (which is "dbabcba"). We can see that i -right palindrome (at position 15) can't expand more than i -left palindrome (at position 7).

If there was a possibility of expansion, i -left palindrome could have expanded itself more already. But there is no such possibility as i -left palindrome is prefix of center palindrome. So due to symmetry property, i -right palindrome will be exactly same as i -left palindrome and it can't expand more. This makes $L[\text{currentRightPosition}] = L[\text{currentLeftPosition}]$ in Case 1.

Now if we consider $\text{centerPosition} = 19$, then $\text{centerLeftPosition} = 12$ and $\text{centerRightPosition} = 26$

If we take $\text{currentRightPosition} = 23$, it's $\text{currentLeftPosition}$ is 15. Case 2 applies here and so $L[23] = 3$. i -left palindrome at position 15 is "bab" which is completely contained in center palindrome at position 19 (which is "babdbab"). In Case 2, where i -left palindrome is prefix of center palindrome, i -right palindrome can't expand more than length of i -left palindrome because center palindrome is suffix of input string so there are no more character left to compare and expand. This makes $L[\text{currentRightPosition}] = L[\text{currentLeftPosition}]$ in Case 2.

Case 1: $L[\text{currentRightPosition}] = L[\text{currentLeftPosition}]$ applies when:

- i -left palindrome is completely contained in center palindrome

- i-left palindrome is NOT a prefix of center palindrome

Above above conditions are satisfied when

$$L[\text{currentLeftPosition}] < \text{centerRightPosition} - \text{currentRightPosition}$$

Case 2: $L[\text{currentRightPosition}] = L[\text{currentLeftPosition}]$ applies when:

- i-left palindrome is prefix of center palindrome (means completely contained also)
- center palindrome is suffix of input string

Above conditions are satisfied when

$$L[\text{currentLeftPosition}] = \text{centerRightPosition} - \text{currentRightPosition} \text{ (For 1st condition) AND}$$

$$\text{centerRightPosition} = 2*N \text{ where } N \text{ is input string length } N \text{ (For 2nd condition).}$$

Case 3: $L[\text{currentRightPosition}] >= L[\text{currentLeftPosition}]$ applies when:

- i-left palindrome is prefix of center palindrome (and so i-left palindrome is completely contained in center palindrome)
- center palindrome is NOT suffix of input string

Above conditions are satisfied when

$$L[\text{currentLeftPosition}] = \text{centerRightPosition} - \text{currentRightPosition} \text{ (For 1st condition) AND}$$

$$\text{centerRightPosition} < 2*N \text{ where } N \text{ is input string length } N \text{ (For 2nd condition).}$$

In this case, there is a possibility of i-right palindrome expansion and so length of i-right palindrome is at least as long as length of i-left palindrome.

Case 4: $L[\text{currentRightPosition}] >= \text{centerRightPosition} - \text{currentRightPosition}$ applies when:

- i-left palindrome is NOT completely contained in center palindrome

Above condition is satisfied when

$$L[\text{currentLeftPosition}] > \text{centerRightPosition} - \text{currentRightPosition}$$

In this case, length of i-right palindrome is at least as long ($\text{centerRightPosition} - \text{currentRightPosition}$) and there is a possibility of i-right palindrome expansion.

In following figure,

(click to see it clearly)																													
String S		b		a		b		c		b		a		b		c		b		a		b		a					
LPS Length L	0	1	0	3	0	1	0	7	0	1	0	9	0	1	0	3	0	1	0	1	0	1	2	1	0	1			
Position I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

If we take center position 7, then Case 3 applies at currentRightPosition 11 because i-left palindrome at currentLeftPosition 3 is a prefix of center palindrome and i-right palindrome is not suffix of input string, so here $L[11] = 9$, which is greater than i-left palindrome length $L[3] = 3$. In the case, it is guaranteed that $L[11]$ will be at least 3, and so in implementation, we 1st set $L[11] = 3$ and then we try to expand it by comparing characters in left and right side starting from distance 4 (As up-to distance 3, it is already known that characters will match).

If we take center position 11, then Case 4 applies at currentRightPosition 15 because $L[\text{currentLeftPosition}] = L[7] = 7 > \text{centerRightPosition} - \text{currentRightPosition} = 20 - 15 = 5$. In the case, it is guaranteed that $L[15]$ will be at least 5, and so in implementation, we 1st set $L[15] = 5$ and then we try to expand it by comparing characters in left and right side starting from distance 5 (As up-to distance 5, it is already known that characters will match).

Now one point left to discuss is, when we work at one center position and compute LPS lengths for different rightPositions, how to know that what would be next center position. We change centerPosition to currentRightPosition if palindrome centered at currentRightPosition expands beyond centerRightPosition.

Here we have seen four different cases on how LPS length of a position will depend on a previous position's LPS length.

In [Part 3](#), we have discussed code implementation of it and also we have looked at these four cases in a different way and implement that too.

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



Ali Tile BY **LOLA GROUP**

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Pattern Searching Strings palindrome

Related Posts:

- Wildcard Pattern Matching
- Find all occurrences of a given word in a matrix
- Aho-Corasick Algorithm for Pattern Searching
- [kasai's Algorithm for Construction of LCP array from Suffix Array](#)
- Search a Word in a 2D Grid of characters
- [Z algorithm \(Linear time pattern searching Algorithm\)](#)
- Suffix Tree Application 6 – Longest Palindromic Substring
- [Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4](#)

(Login to Rate and Mark)

4.1

Average Difficulty : 4.1/5.0
Based on 12 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

@geeksforgeeks, Some rights reserved

[Contact Us!](#) [About Us!](#)
[Privacy Policy](#)

[Advertise with us!](#)

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 3

In Manacher's Algorithm [Part 1](#) and [Part 2](#), we gone through some of the basics, understood LPS length array and how to calculate it efficiently based on four cases. Here we will implement the same.

We have seen that there are no new character comparison needed in case 1 and case 2. In case 3 and case 4, necessary new comparison are needed.

In following figure,

(click to see it clearly)																													
String S		b		a		b		c		b		a		b		c		b		a		b		a					
LPS Length L	0	1	0	3	0	1	0	7	0	1	0	9	0	1	0	3	0	1	0	1	0	1	2	1	0	1	0		
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

If at all we need a comparison, we will only compare actual characters, which are at "odd" positions like 1, 3, 5, 7, etc.

Even positions do not represent a character in string, so no comparison will be preformed for even positions.

If two characters at different odd positions match, then they will increase LPS length by 2.

There are many ways to implement this depending on how even and odd positions are handled. One way would be to create a new string 1st where we insert some unique character (say #, \$ etc) in all even positions and then run algorithm on that (to avoid different way of even and odd position handling). Other way could be to work on given string itself but here even and odd positions should be handled appropriately.

Here we will start with given string itself. When there is a need of expansion and character comparison required, we will expand in left and right positions one by one. When odd position is found, comparison will be done and LPS Length will be incremented by ONE. When even position is found, no comparison done and LPS Length will be incremented by ONE (So overall, one odd and one even positions on both left and right side will increase LPS Length by TWO).

```
// A C program to implement Manacher's Algorithm
#include <stdio.h>
#include <string.h>

char text[100];
void findLongestPalindromicString()
{
    int N = strlen(text);
    if(N == 0)
        return;
```

```

N = 2*N + 1; //Position count
int L[N]; //LPS Length Array
L[0] = 0;
L[1] = 1;
int C = 1; //centerPosition
int R = 2; //centerRightPosition
int i = 0; //currentRightPosition
int iMirror; //currentLeftPosition
int expand = -1;
int diff = -1;
int maxLPSLength = 0;
int maxLPSCenterPosition = 0;
int start = -1;
int end = -1;

//Uncomment it to print LPS Length array
//printf("%d %d ", L[0], L[1]);
for (i = 2; i < N; i++)
{
    //get currentLeftPosition iMirror for currentRightPosition i
    iMirror = 2*C-i;
    //Reset expand - means no expansion required
    expand = 0;
    diff = R - i;
    //If currentRightPosition i is within centerRightPosition R
    if(diff > 0)
    {
        if(L[iMirror] < diff) // Case 1
            L[i] = L[iMirror];
        else if(L[iMirror] == diff && i == N-1) // Case 2
            L[i] = L[iMirror];
        else if(L[iMirror] == diff && i < N-1) // Case 3
        {
            L[i] = L[iMirror];
            expand = 1; // expansion required
        }
        else if(L[iMirror] > diff) // Case 4
        {
            L[i] = diff;
            expand = 1; // expansion required
        }
    }
    else
    {
        L[i] = 0;
        expand = 1; // expansion required
    }

    if (expand == 1)
    {
        //Attempt to expand palindrome centered at currentRightPosition i
        //Here for odd positions, we compare characters and
        //if match then increment LPS Length by ONE
        //If even position, we just increment LPS by ONE without
        //any character comparison
        while (((i + L[i]) < N && (i - L[i]) > 0) &&
               ( ((i + L[i] + 1) % 2 == 0) ||
                 (text[(i + L[i] + 1)/2] == text[(i-L[i]-1)/2] )))
        {
            L[i]++;
        }
    }

    if(L[i] > maxLPSLength) // Track maxLPSLength
    {
        maxLPSLength = L[i];
        maxLPSCenterPosition = i;
    }

    // If palindrome centered at currentRightPosition i
    // expand beyond centerRightPosition R,
    // adjust centerPosition C based on expanded palindrome.
    if (i + L[i] > R)
    {

```

```

        C = i;
        R = i + L[i];
    }
    //Uncomment it to print LPS Length array
    //printf("%d ", L[i]);
}
//printf("\n");
start = (maxLPSCenterPosition - maxLPSLength)/2;
end = start + maxLPSLength - 1;
//printf("start: %d end: %d\n", start, end);
printf("LPS of string is %s : ", text);
for(i=start; i<=end; i++)
    printf("%c", text[i]);
printf("\n");
}

int main(int argc, char *argv[])
{
    strcpy(text, "babcbabcaccba");
    findLongestPalindromicString();

    strcpy(text, "abaaba");
    findLongestPalindromicString();

    strcpy(text, "abababa");
    findLongestPalindromicString();

    strcpy(text, "abcbabcbabcba");
    findLongestPalindromicString();

    strcpy(text, "forgeeksskeegfor");
    findLongestPalindromicString();

    strcpy(text, "caba");
    findLongestPalindromicString();

    strcpy(text, "abacdfgdcaba");
    findLongestPalindromicString();

    strcpy(text, "abacdfgdcabba");
    findLongestPalindromicString();

    strcpy(text, "abacdedcaba");
    findLongestPalindromicString();

    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to implement Manacher's Algorithm

def findLongestPalindromicString(text):
    N = len(text)
    if N == 0:
        return
    N = 2*N+1      # Position count
    L = [0] * N
    L[0] = 0
    L[1] = 1
    C = 1          # centerPosition
    R = 2          # centerRightPosition
    i = 0          # currentRightPosition
    iMirror = 0     # currentLeftPosition
    maxLPSLength = 0
    maxLPSCenterPosition = 0
    start = -1

```

```

end = -1
diff = -1

# Uncomment it to print LPS Length array
# printf("%d %d ", L[0], L[1]);
for i in xrange(2,N):

    # get currentLeftPosition iMirror for currentRightPosition i
    iMirror = 2*C-i
    L[i] = 0
    diff = R - i
    # If currentRightPosition i is within centerRightPosition R
    if diff > 0:
        L[i] = min(L[iMirror], diff)

    # Attempt to expand palindrome centered at currentRightPosition i
    # Here for odd positions, we compare characters and
    # if match then increment LPS Length by ONE
    # If even position, we just increment LPS by ONE without
    # any character comparison
    try:
        while ((i+L[i]) < N and (i-L[i]) > 0) and \
            (((i+L[i]+1) % 2 == 0) or \
            (text[(i+L[i]+1)/2] == text[(i-L[i]-1)/2])):
            L[i] += 1
    except Exception as e:
        pass

    if L[i] > maxLPSLength:           # Track maxLPSLength
        maxLPSLength = L[i]
        maxLPSCenterPosition = i

    # If palindrome centered at currentRightPosition i
    # expand beyond centerRightPosition R,
    # adjust centerPosition C based on expanded palindrome.
    if i + L[i] > R:
        C = i
        R = i + L[i]

# Uncomment it to print LPS Length array
# printf("%d ", L[i]);
start = (maxLPSCenterPosition - maxLPSLength) / 2
end = start + maxLPSLength - 1
print "LPS of string is " + text + " : ",
print text[start:end+1],
print "\n",

# Driver program
text1 = "babcbabcba"
findLongestPalindromicString(text1)

text2 = "abaaba"
findLongestPalindromicString(text2)

text3 = "abababa"
findLongestPalindromicString(text3)

text4 = "abcbabcbabcba"
findLongestPalindromicString(text4)

text5 = "forgeeksskeegfor"
findLongestPalindromicString(text5)

text6 = "caba"
findLongestPalindromicString(text6)

text7 = "abacdfgdcaba"
findLongestPalindromicString(text7)

text8 = "abacdfgdcabba"
findLongestPalindromicString(text8)

text9 = "abacdedcba"
findLongestPalindromicString(text9)

```

This code is contributed by BHAVYA JAIN

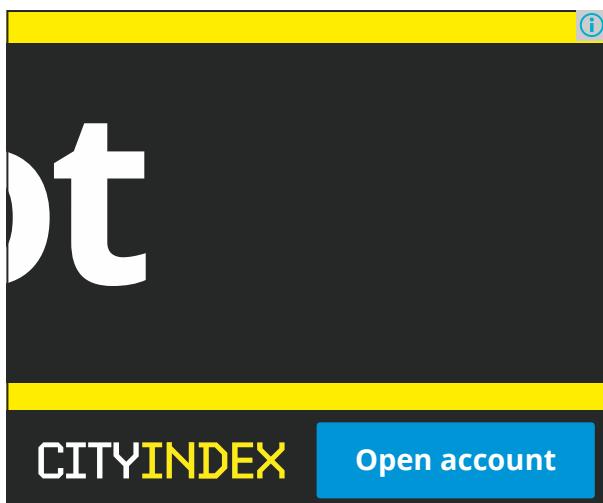
[Run on IDE](#)

Output:

```
LPS of string is babcbabcaccba : abcbabcba
LPS of string is abaaba : abaaba
LPS of string is abababa : abababa
LPS of string is abcbabcbabcba : abcbabcbabcba
LPS of string is forgeeksskeegfor : geeksskeeg
LPS of string is caba : aba
LPS of string is abacdfgdcaba : aba
LPS of string is abacdfgdcabba : abba
LPS of string is abacdedcaba : abacdedcaba
```

This is the implementation based on the four cases discussed in [Part 2](#). In [Part 4](#), we have discussed a different way to look at these four cases and few other approaches.

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



GATE CS Corner Company Wise Coding Practice

[Pattern Searching](#) [Strings](#) [palindrome](#)

Related Posts:

- [Wildcard Pattern Matching](#)
- [Find all occurrences of a given word in a matrix](#)
- [Aho-Corasick Algorithm for Pattern Searching](#)
- [Kasai's Algorithm for Construction of LCP array from Suffix Array](#)
- [Search a Word in a 2D Grid of characters](#)

- Z algorithm (Linear time pattern searching Algorithm)
- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

(Login to Rate and Mark)

4.5

Average Difficulty : 4.5/5.0
Based on 10 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#) [GATE CS](#) [Placements](#) [GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

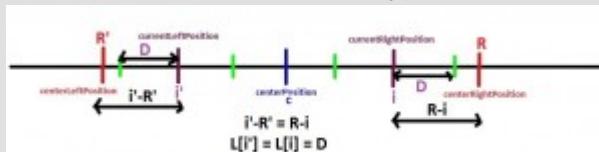
Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 4

In Manacher's Algorithm [Part 1](#) and [Part 2](#), we gone through some of the basics, understood LPS length array and how to calculate it efficiently based on four cases. In [Part 3](#), we implemented the same.

Here we will review the four cases again and try to see it differently and implement the same.

All four cases depends on LPS length value at currentLeftPosition ($L[iMirror]$) and value of ($centerRightPosition - currentRightPosition$), i.e. ($R - i$). These two information are know before which helps us to reuse previous available information and avoid unnecessary character comparison.

(click to see it clearly)



If we look at all four cases, we will see that we 1st set minimum of $L[iMirror]$ and $R-i$ to $L[i]$ and then we try to expand the palindrome in whichever case it can expand.

Above observation may look more intuitive, easier to understand and implement, given that one understands LPS length array, position, index, symmetry property etc.

```
// A C program to implement Manacher's Algorithm
#include <stdio.h>
#include <string.h>

char text[100];
int min(int a, int b)
{
    int res = a;
    if(b < a)
        res = b;
    return res;
}

void findLongestPalindromicString()
{
    int N = strlen(text);
    if(N == 0)
        return;
    N = 2*N + 1; //Position count
    int L[N]; //LPS Length Array
    L[0] = 0;
```

```

L[1] = 1;
int C = 1; //centerPosition
int R = 2; //centerRightPosition
int i = 0; //currentRightPosition
int iMirror; //currentLeftPosition
int maxLPSLength = 0;
int maxLPSCenterPosition = 0;
int start = -1;
int end = -1;
int diff = -1;

//Uncomment it to print LPS Length array
//printf("%d %d ", L[0], L[1]);
for (i = 2; i < N; i++)
{
    //get currentLeftPosition iMirror for currentRightPosition i
    iMirror = 2*C-i;
    L[i] = 0;
    diff = R - i;
    //If currentRightPosition i is within centerRightPosition R
    if(diff > 0)
        L[i] = min(L[iMirror], diff);

    //Attempt to expand palindrome centered at currentRightPosition i
    //Here for odd positions, we compare characters and
    //if match then increment LPS Length by ONE
    //If even position, we just increment LPS by ONE without
    //any character comparison
    while ( ((i + L[i]) < N && (i - L[i]) > 0) &&
           (((i + L[i] + 1) % 2 == 0) ||
            (text[(i + L[i] + 1)/2] == text[(i - L[i] - 1)/2])))
    {
        L[i]++;
    }

    if(L[i] > maxLPSLength) // Track maxLPSLength
    {
        maxLPSLength = L[i];
        maxLPSCenterPosition = i;
    }

    //If palindrome centered at currentRightPosition i
    //expand beyond centerRightPosition R,
    //adjust centerPosition C based on expanded palindrome.
    if (i + L[i] > R)
    {
        C = i;
        R = i + L[i];
    }
    //Uncomment it to print LPS Length array
    //printf("%d ", L[i]);
}
//printf("\n");
start = (maxLPSCenterPosition - maxLPSLength)/2;
end = start + maxLPSLength - 1;
printf("LPS of string is %s : ", text);
for(i=start; i<=end; i++)
    printf("%c", text[i]);
printf("\n");
}

int main(int argc, char *argv[])
{
    strcpy(text, "babcbabcbaccba");
    findLongestPalindromicString();

    strcpy(text, "abaaba");
    findLongestPalindromicString();

    strcpy(text, "abababa");
    findLongestPalindromicString();

    strcpy(text, "abcbabcbabcb");
}

```

```

findLongestPalindromicString();

strcpy(text, "forgeeksskeegfor");
findLongestPalindromicString();

strcpy(text, "caba");
findLongestPalindromicString();

strcpy(text, "abacdfgdcaba");
findLongestPalindromicString();

strcpy(text, "abacdfgdcabba");
findLongestPalindromicString();

strcpy(text, "abacdedcaba");
findLongestPalindromicString();

return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to implement Manacher's Algorithm

def findLongestPalindromicString(text):
    N = len(text)
    if N == 0:
        return
    N = 2*N+1      # Position count
    L = [0] * N
    L[0] = 0
    L[1] = 1
    C = 1          # centerPosition
    R = 2          # centerRightPosition
    i = 0          # currentRightPosition
    iMirror = 0     # currentLeftPosition
    maxLPSLength = 0
    maxLPSCenterPosition = 0
    start = -1
    end = -1
    diff = -1

    # Uncomment it to print LPS Length array
    # printf("%d %d ", L[0], L[1]);
    for i in xrange(2,N):

        # get currentLeftPosition iMirror for currentRightPosition i
        iMirror = 2*C-i
        L[i] = 0
        diff = R - i
        # If currentRightPosition i is within centerRightPosition R
        if diff > 0:
            L[i] = min(L[iMirror], diff)

        # Attempt to expand palindrome centered at currentRightPosition i
        # Here for odd positions, we compare characters and
        # if match then increment LPS Length by ONE
        # If even position, we just increment LPS by ONE without
        # any character comparison
        try:
            while ((i + L[i]) < N and (i - L[i]) > 0) and \
                (((i + L[i] + 1) % 2 == 0) or \
                 (text[(i + L[i] + 1) / 2] == text[(i - L[i] - 1) / 2])):
                L[i]+=1
        except Exception as e:
            pass

        if L[i] > maxLPSLength:      # Track maxLPSLength
            maxLPSLength = L[i]

```

```

maxLPSCenterPosition = i

# If palindrome centered at currentRightPosition i
# expand beyond centerRightPosition R,
# adjust centerPosition C based on expanded palindrome.
if i + L[i] > R:
    C = i
    R = i + L[i]

# Uncomment it to print LPS Length array
# printf("%d ", L[i]);
start = (maxLPSCenterPosition - maxLPSLength) / 2
end = start + maxLPSLength - 1
print "LPS of string is " + text + " : ",
print text[start:end+1],
print "\n",

# Driver program
text1 = "babcbabcaccba"
findLongestPalindromicString(text1)

text2 = "abaaba"
findLongestPalindromicString(text2)

text3 = "abababa"
findLongestPalindromicString(text3)

text4 = "abcbabcbabcba"
findLongestPalindromicString(text4)

text5 = "forgeeksskeegfor"
findLongestPalindromicString(text5)

text6 = "caba"
findLongestPalindromicString(text6)

text7 = "abacdfgdcaba"
findLongestPalindromicString(text7)

text8 = "abacdfgdcabba"
findLongestPalindromicString(text8)

text9 = "abacdedcaba"
findLongestPalindromicString(text9)

# This code is contributed by BHAVYA JAIN

```

[Run on IDE](#)

Output:

```

LPS of string is babcbabcaccba : abcbabcba
LPS of string is abaaba : abaaba
LPS of string is abababa : abababa
LPS of string is abcbabcbabcba : abcbabcbabcba
LPS of string is forgeeksskeegfor : geeksskeeg
LPS of string is caba : aba
LPS of string is abacdfgdcaba : aba
LPS of string is abacdfgdcabba : abba
LPS of string is abacdedcaba : abacdedcaba

```

Other Approaches

We have discussed two approaches here. One in [Part 3](#) and other in current article. In both approaches, we worked on given string. Here we had to handle even and odd positions differently while comparing characters for expansion (because even positions do not represent any character in string).

To avoid this different handling of even and odd positions, we need to make even positions also to represent some character (actually all even positions should represent SAME character because they MUST match while character comparison). One way to do this is to set some character at all even positions by modifying given string or create a new copy of given string. For example, if input string is "abcb", new string should be "#a#b#c#b#" if we add # as unique character at even positions.

The two approaches discussed already can be modified a bit to work on modified string where different handling of even and odd positions will not be needed.

We may also add two DIFFERENT characters (not yet used anywhere in string at even and odd positions) at start and end of string as sentinels to avoid bound check. With these changes string "abcb" will look like "^#a#b#c#b#\$" where ^ and \$ are sentinels.

This implementation may look cleaner with the cost of more memory.

We are not implementing these here as it's a simple change in given implementations.

Implementation of approach discussed in current article on a modified string can be found at [Longest Palindromic Substring Part II](#) and a [Java Translation](#) of the same by Princeton.

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

Alitile
BY LOLA GROUP

Address: 2910, Jalan Petaling 24/1, Surya Commerce, 4710 Petaling Jaya, Malaysia

China huge tile factory

Huge tile factory, 25 years exporting experiences, branches in USA, India, Australia, Malaysia

alitile.com

GATE CS Corner Company Wise Coding Practice

Pattern Searching | Strings | palindrome

Related Posts:

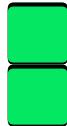
- [Wildcard Pattern Matching](#)
- [Find all occurrences of a given word in a matrix](#)
- [Aho-Corasick Algorithm for Pattern Searching](#)
- [Kasai's Algorithm for Construction of LCP array from Suffix Array](#)
- [Search a Word in a 2D Grid of characters](#)
- [Z algorithm \(Linear time pattern searching Algorithm\)](#)

- Suffix Tree Application 6 – Longest Palindromic Substring
- Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 3

(Login to Rate and Mark)

5

Average Difficulty : 5/5.0
Based on 12 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use [code.geeksforgeeks.org](#), generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Print all possible strings that can be made by placing spaces

Given a string you need to print all possible strings that can be made by placing spaces (zero or one) in between them.

Input: str[] = "ABC"

Output: ABC

AB C

A BC

A B C

Source: Amazon Interview Experience | Set 158, Round 1 ,Q 1.

We strongly recommend that you click here and practice it, before moving on to the solution.

The idea is to use recursion and create a buffer that one by one contains all output strings having spaces. We keep updating buffer in every recursive call. If the length of given string is 'n' our updated string can have maximum length of $n + (n-1)$ i.e. $2n-1$. So we create buffer size of $2n$ (one extra character for string termination). We leave 1st character as it is, starting from the 2nd character, we can either fill a space or a character. Thus one can write a recursive function like below.

```
// C++ program to print permutations of a given string with spaces.
#include <iostream>
#include <cstring>
using namespace std;

/* Function recursively prints the strings having space pattern.
 i and j are indices in 'str[]' and 'buff[]' respectively */
void printPatternUtil(char str[], char buff[], int i, int j, int n)
{
    if (i==n)
    {
        buff[j] = '\0';
        cout << buff << endl;
        return;
    }

    // Either put the character
    buff[j] = str[i];
```

```

printPatternUtil(str, buff, i+1, j+1, n);

// Or put a space followed by next character
buff[j] = ' ';
buff[j+1] = str[i];

printPatternUtil(str, buff, i+1, j+2, n);
}

// This function creates buf[] to store individual output string and uses
// printPatternUtil() to print all permutations.
void printPattern(char *str)
{
    int n = strlen(str);

    // Buffer to hold the string containing spaces
    char buf[2*n]; // 2n-1 characters and 1 string terminator

    // Copy the first character as it is, since it will be always
    // at first position
    buf[0] = str[0];

    printPatternUtil(str, buf, 1, 1, n);
}

// Driver program to test above functions
int main()
{
    char *str = "ABCD";
    printPattern(str);
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to print permutations of a given string with
# spaces.

# Utility function
def toString(List):
    s = ""
    for x in List:
        if x == '\0':
            break
        s += x
    return s

# Function recursively prints the strings having space pattern.
# i and j are indices in 'str[]' and 'buff[]' respectively
def printPatternUtil(string, buff, i, j, n):
    if i == n:
        buff[j] = '\0'
        print toString(buff)
        return

    # Either put the character
    buff[j] = string[i]
    printPatternUtil(string, buff, i+1, j+1, n)

    # Or put a space followed by next character
    buff[j] = ' '
    buff[j+1] = string[i]

    printPatternUtil(string, buff, i+1, j+2, n)

# This function creates buf[] to store individual output string
# and uses printPatternUtil() to print all permutations.
def printPattern(string):
    n = len(string)

```

```

# Buffer to hold the string containing spaces
buff = [0] * (2*n) # 2n-1 characters and 1 string terminator

# Copy the first character as it is, since it will be always
# at first position
buff[0] = string[0]

printPatternUtil(string, buff, 1, 1, n)

# Driver program
string = "ABCD"
printPattern(string)

# This code is contributed by BHAVYA JAIN

```

[Run on IDE](#)

Output:

```

ABCD
ABC D
AB CD
AB C D
A BCD
A BC D
A B CD
A B C D

```

Time Complexity: Since number of Gaps are $n-1$, there are total 2^{n-1} patterns each having length ranging from n to $2n-1$. Thus overall complexity would be $O(n*(2^n))$.

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



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Strings

Related Posts:

- Nth character in Concatenated Decimal String
- Convert to a string that is repetition of a substring of k length
- Minimum characters to be added at front to make string palindrome
- Count All Palindrome Sub-Strings in a String
- Check for Palindrome after every character replacement Query
- Group all occurrences of characters according to first appearance
- Count characters at same position as in English alphabets
- Find if an array of strings can be chained to form a circle | Set 2

(Login to Rate and Mark)

3.3

Average Difficulty : **3.3/5.0**
Based on **49** vote(s)



Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!

GeeksforGeeks

A computer science portal for geeks

[Practice](#)

[GATE CS](#)

[Placements](#)

[GeeksQuiz](#)

Google™ Custom Search



[Login/Register](#)

Write a program to print all permutations of a given string

A permutation, also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has $n!$ permutation.

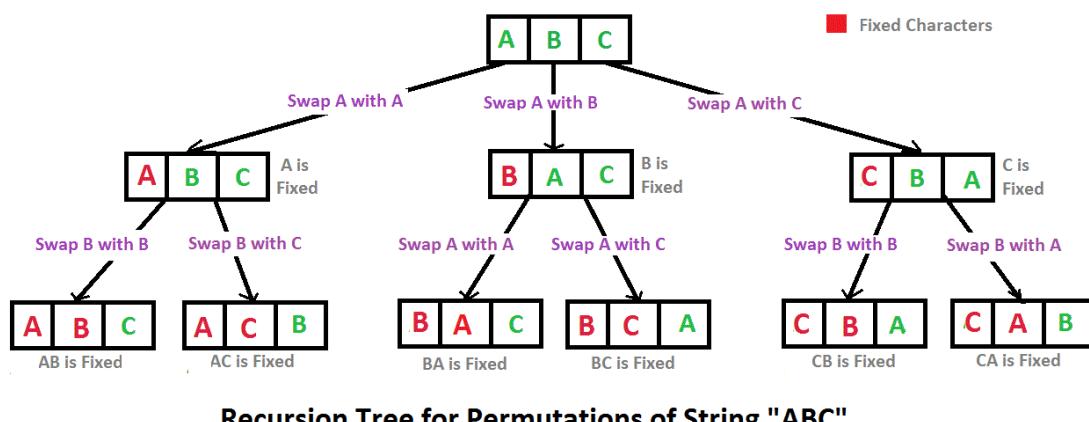
Source: Mathword(<http://mathworld.wolfram.com/Permutation.html>)

Below are the permutations of string ABC.

ABC ACB BAC BCA CBA CAB

We strongly recommend that you click here and practice it, before moving on to the solution.

Here is a solution that is used as a basis in backtracking.



```
// C program to print all permutations with duplicates allowed
#include <stdio.h>
#include <string.h>

/* Function to swap values at two pointers */
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

```

*x = *y;
*y = temp;
}

/* Function to print permutations of string
This function takes three parameters:
1. String
2. Starting index of the string
3. Ending index of the string. */
void permute(char *a, int l, int r)
{
    int i;
    if (l == r)
        printf("%s\n", a);
    else
    {
        for (i = l; i <= r; i++)
        {
            swap((a+l), (a+i));
            permute(a, l+1, r);
            swap((a+l), (a+i)); //backtrack
        }
    }
}

/* Driver program to test above functions */
int main()
{
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}

```

[Run on IDE](#)

Python

```

# Python program to print all permutations with
# duplicates allowed

def toString(List):
    return ''.join(List)

# Function to print permutations of string
# This function takes three parameters:
# 1. String
# 2. Starting index of the string
# 3. Ending index of the string.
def permute(a, l, r):
    if l==r:
        print toString(a)
    else:
        for i in xrange(l,r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l] # backtrack

# Driver program to test the above function
string = "ABC"
n = len(string)
a = list(string)
permute(a, 0, n-1)

# This code is contributed by Bhavya Jain

```

[Run on IDE](#)

Output:

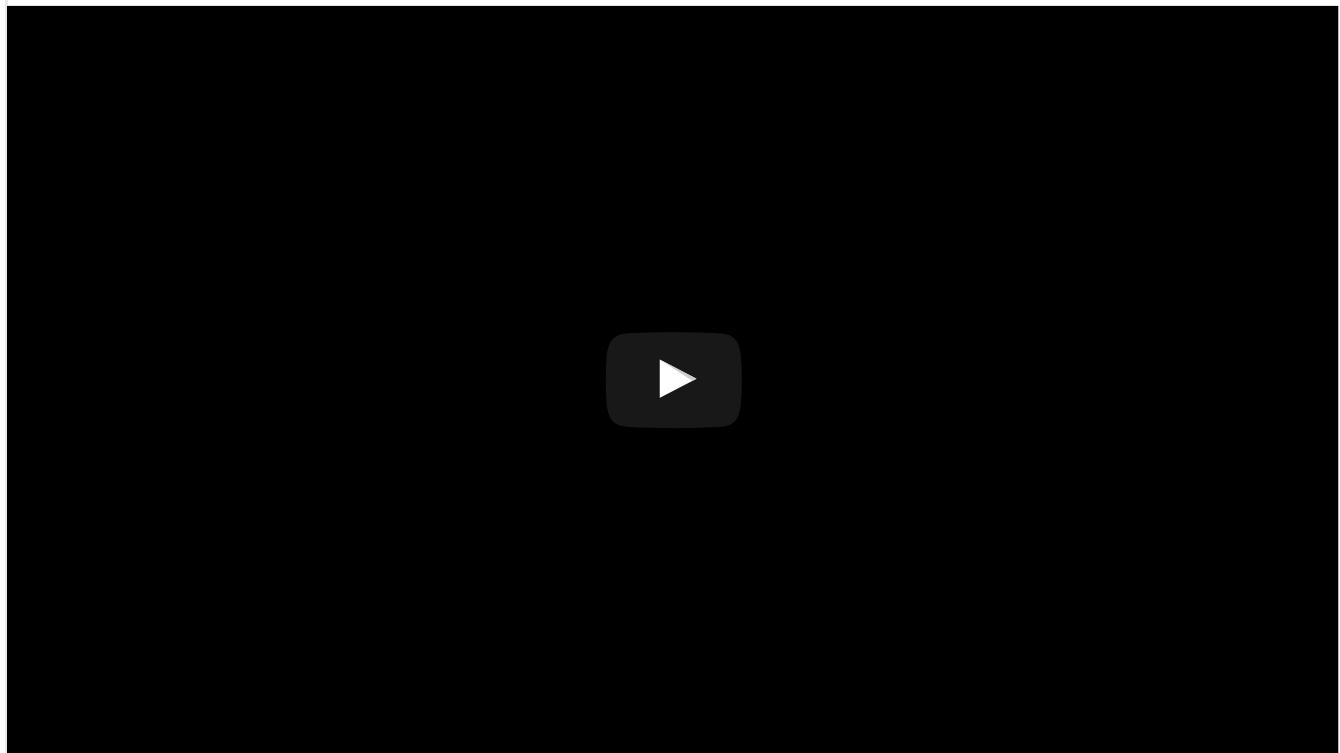
```
ABC  
ACB  
BAC  
BCA  
CBA  
CAB
```

Algorithm Paradigm: Backtracking

Time Complexity: $O(n*n!)$ Note that there are $n!$ permutations and it requires $O(n)$ time to print a permutation.

Note : The above solution prints duplicate permutations if there are repeating characters in input string. Please see below link for a solution that prints only distinct permutations even if there are duplicates in input.

[Print all distinct permutations of a given string with duplicates.](#)



Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.



Download

Free Download

unzipper.com



GATE CS Corner Company Wise Coding Practice

Combinatorial Strings Backtracking MathematicalAlgo permutation

Related Posts:

- Sum of all numbers that can be formed with permutations of n digits
- Longest common subsequence with permutations allowed
- Number of ways to make mobile lock pattern
- Count permutations that produce positive result
- Find all distinct subsets of a given set
- Heap's Algorithm for generating permutations
- All permutations of a string using iteration
- Permutations of a given string using STL

(Login to Rate and Mark)

3.4

Average Difficulty : 3.4/5.0
Based on 180 vote(s)



Add to TODO List



Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments

@geeksforgeeks, Some rights reserved

Contact Us! About Us!
Privacy Policy

Advertise with us!