Contents

Contents

Chapter 1

Activity Selection Problem Greedy Algo-1

Activity Selection Problem Greedy Algo-1 - GeeksforGeeks

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: a 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: a 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 optimiza-

tion 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 don't always produce the best optimal solution but can be used to get an approximately 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:

```
Example 1 : Consider the following 3 activities sorted by
by finish time.
     start[]  =  {10, 12, 20};
     finish[] =  {20, 25, 30};
A person can perform at most two activities. The
maximum set of activities that can be executed
is {0, 2} [ These are indexes in start[] and
finish[] ]

Example 2 : Consider the following 6 activities
sorted by by finish time.
     start[]  =  {1, 3, 0, 5, 8, 5};
     finish[] =  {2, 4, 6, 7, 9, 9};
A person can perform at most four activities. The
maximum set of activities that can be executed
is {0, 1, 3, 4} [ These are indexes in start[] and
finish[] ]
```

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 or equal to 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.

C++

```
 // C++ program for activity selection problem.
// The following implementation assumes 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);
    return 0;
}
```

Java

```
 // The following implementation assumes that the activities
// are already sorted according to their finish time
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);
    }

}
```

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 contains 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
```

Output:

```
Following activities are selected
0 1 3 4
```

How does Greedy Choice work for Activities sorted according to finish time?
Let the given set of activities be S = {1, 2, 3, ..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 the first activity other than 1, then there is also a solution A of the same size with activity 1 as the first activity. Let the first activity selected by B be k, then there always exist A = {B – {k}} U {1}.(Note that the activities in B are independent and k has smallest finishing time among all. Since k is not 1, finish(k) >= finish(1)).

How to implement when given activities are not sorted?
We create a structure/class for activities. We sort all activities by finish time (Refer sort in C++ STL). Once we have activities sorted, we apply same above algorithm.

```
 // C++ program for activity selection problem
// when input activities may not be sorted.
#include <bits/stdc++.h>
using namespace std;

// A job has a start time, finish time and profit.
struct Activitiy
{
    int start, finish;
};

// A utility function that is used for sorting
// activities according to finish time
bool activityCompare(Activitiy s1, Activitiy s2)
{
    return (s1.finish < s2.finish);
}

// Returns count of the maximum set of activities that can
// be done by a single person, one at a time.
void printMaxActivities(Activitiy arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, activityCompare);

    cout << "Following activities are selected n";

    // The first activity always gets selected
    int i = 0;
    cout << "(" << arr[i].start << ", " << arr[i].finish << "), ";

    // Consider rest of the activities
    for (int 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 (arr[j].start >= arr[i].finish)
      {
          cout << "(" << arr[j].start << ", "
              << arr[j].finish << "), ";
          i = j;
      }
    }
```

```
}

// Driver program
int main()
{
    Activitiy arr[] = {{5, 9}, {1, 2}, {3, 4}, {0, 6},
                                       {5, 7}, {8, 9}};
    int n = sizeof(arr)/sizeof(arr[0]);
    printMaxActivities(arr, n);
    return 0;
}
```

Output:

```
Following activities are selected
(1, 2), (3, 4), (5, 7), (8, 9),
```

Time Complexity : It takes O(n log n) time if input activities may not be sorted. It takes O(n) time when it is given that input activities are always sorted.

References:
Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
http://en.wikipedia.org/wiki/Greedy_algorithm

Source

https://www.geeksforgeeks.org/activity-selection-problem-greedy-algo-1/

Chapter 2

Applications of Minimum Spanning Tree Problem

Applications of Minimum Spanning Tree Problem - GeeksforGeeks

Minimum Spanning Tree (MST) problem: Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.

MST is fundamental problem with diverse applications.

Network design.
– *telephone, electrical, hydraulic, TV cable, computer, road*
The standard application is to a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money.

Approximation algorithms for NP-hard problems.
– *traveling salesperson problem, Steiner tree*
A less obvious application is that the minimum spanning tree can be used to approximately solve the traveling salesman problem. A convenient formal way of defining this problem is to find the shortest path that visits each point at least once.

Note that if you have a path visiting all points exactly once, it's a special kind of tree. For instance in the example above, twelve of sixteen spanning trees are actually paths. If you have a path visiting some vertices more than once, you can always drop some edges to get a tree. So in general the MST weight is less than the TSP weight, because it's a minimization over a strictly larger set.

On the other hand, if you draw a path tracing around the minimum spanning tree, you trace each edge twice and visit all points, so the TSP weight is less than twice the MST weight. Therefore this tour is within a factor of two of optimal.

Indirect applications.
– max bottleneck paths
– LDPC codes for error correction
– image registration with Renyi entropy
– learning salient features for real-time face verification
– reducing data storage in sequencing amino acids in a protein
– model locality of particle interactions in turbulent fluid flows
– autoconfig protocol for Ethernet bridging to avoid cycles in a network

Cluster analysis
k clustering problem can be viewed as finding an MST and deleting the k-1 most expensive edges.

Sources:
http://www.cs.princeton.edu/courses/archive/spr07/cos226/lectures/mst.pdf
http://www.ics.uci.edu/~eppstein/161/960206.html

Source

https://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/

Chapter 3

Array element moved by k using single moves

Array element moved by k using single moves - GeeksforGeeks

Given a list of n integers containing numbers 1-n in a shuffled way and a integer K. N people are standing in a queue to play badminton. At first, the first two players in the queue play a game. Then the loser goes to the end of the queue, and the one who wins plays with the next person from the line, and so on. They play until someone wins k games consecutively. This player becomes the winner.

Examples :

```
Input: arr[] = {2, 1, 3, 4, 5}
       k = 2
Output: 5
Explanation:
2 plays with 1, 1 goes to end of queue.
2 plays with 3, 3 wins, 2 goes to end of queue.
3 plays with 4, so 3 goes to the end of the queue.
5 plays with everyone and wins as it is the
largest of all elements.

Input: arr[] = {3, 1, 2}
       k = 2
Output: 3
Explanation :
3 plays with 1. 3 wins. 1 goes to the end of the line.
3 plays with 2. 3 wins. 3 wins twice in a row.
```

A **naive approach** is to run two nested for loops and check for every element which one is more from i to n being the first loop and the second being from i+1 to n and then from 0

to n-1 and count the number of continuous smaller elements and get the answer.
This will not be efficient enough as it takes **O(n*n)** .

An **efficient approach** will be to run a loop from 1 to n and keep track of best (or maximum element) so far and number of smaller elements than this maximum. If current best loose, initialize the greater value to the best and the count to 1, as the winner won 1 time already. If at any step it has won k times, you get your answer. But if k >= n-1, then the maximum number will be the only answer as it will the most number of times being the greatest. If while iterating you don't find any player that has won k times, then the maximum number which is in the list will always be our answer.

Below is the implementation to the above approach

C++

```
 // CPP program to find winner of game
#include <iostream>
using namespace std;

int winner(int a[], int n, int k)
{
    // if the number of steps is more then
    // n-1,
    if (k >= n - 1)
        return n;

    // initially the best is 0 and no of
    // wins is 0.
    int best = 0, times = 0;

    // traverse through all the numbers
    for (int i = 0; i < n; i++) {

        // if the value of array is more
        // then that of previous best
        if (a[i] > best) {

            // best is replaced by a[i]
            best = a[i];

            // if not the first index
            if (i)
                times = 1; // no of wins is 1 now
        }

        else
            times += 1; // if it wins

        // if any position has more then k wins
```

```
        // then return
        if (times >= k)
            return best;
    }

    // Maximum element will be winner because
    // we move smaller element at end and repeat
    // the process.
    return best;
}

// driver program to test the above function
int main()
{
    int a[] = { 2, 1, 3, 4, 5 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 2;
    cout << winner(a, n, k);
    return 0;
}
```

Java

```
 // Java program to find winner of game
import java.io.*;

class GFG {

    static int winner(int a[], int n, int k)
    {
        // if the number of steps is more then
        // n-1,
        if (k >= n - 1)
            return n;

        // initially the best is 0 and no of
        // wins is 0.
        int best = 0, times = 0;

        // traverse through all the numbers
        for (int i = 0; i < n; i++) {

            // if the value of array is more
            // then that of previous best
            if (a[i] > best) {

                // best is replaced by a[i]
                best = a[i];
```

```
                // if not the first index
                if (i == 1)

                    // no of wins is 1 now
                    times = 1;
            }

            else

                // if it wins
                times += 1;

            // if any position has more then
            // k wins then return
            if (times >= k)
                return best;
        }

        // Maximum element will be winner
        // because we move smaller element
        // at end and repeat the process.
        return best;
    }

    // driver program to test the above function
    public static void main(String args[])
    {
        int a[] = { 2, 1, 3, 4, 5 };
        int n = a.length;
        int k = 2;

        System.out.println(winner(a, n, k));
    }
}

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

Python3

```
 # Python code to find
# winner of game

# function to find the winner
def winner( a, n, k):

    # if the number of steps is
    # more then n-1
```

```
    if k >= n - 1:
        return n

    # initially the best is 0
    # and no of wins is 0.
    best = 0
    times = 0

    # traverse through all the numbers
    for i in range(n):

        # if the value of array is more
        # then that of previous best
        if a[i] > best:

            # best is replaced by a[i]
            best = a[i]

            # if not the first index
            if i == True:

                # no of wins is 1 now
                times = 1
        else:
            times += 1 # if it wins

        # if any position has more
        # then k wins then return
        if times >= k:
            return best

    # Maximum element will be winner
    # because we move smaller element
    # at end and repeat the process.
    return best


# driver code
a = [ 2, 1, 3, 4, 5 ]
n = len(a)
k = 2
print(winner(a, n, k))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# program to find winner of game
```

```
using System;

class GFG {

    static int winner(int[] a, int n, int k)
    {

        // if the number of steps is more
        // then n-1,
        if (k >= n - 1)
            return n;

        // initially the best is 0 and no of
        // wins is 0.
        int best = 0, times = 0;

        // traverse through all the numbers
        for (int i = 0; i < n; i++) {

            // if the value of array is more
            // then that of previous best
            if (a[i] > best) {

                // best is replaced by a[i]
                best = a[i];

                // if not the first index
                if (i == 1)

                    // no of wins is 1 now
                    times = 1;
            }

            else

                // if it wins
                times += 1;

            // if any position has more
            // then k wins then return
            if (times >= k)
                return best;
        }

        // Maximum element will be winner
        // because we move smaller element
        // at end and repeat the process.
        return best;
```

```
    }

    // driver program to test the above
    // function
    public static void Main()
    {
        int[] a = { 2, 1, 3, 4, 5 };
        int n = a.Length;
        int k = 2;

        Console.WriteLine(winner(a, n, k));
    }
}

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

PHP

```
 <?php
// PHP program to find winner of game

function winner($a, $n, $k)
{
    // if the number of steps
    // is more then n-1,
    if ($k >= $n - 1)
        return $n;

    // initially the best is 0
    // and no. of wins is 0.
    $best = 0; $times = 0;

    // traverse through all the numbers
    for ($i = 0; $i < $n; $i++)
    {

        // if the value of array is more
        // then that of previous best
        if ($a[$i] > $best)
        {

            // best is replaced by a[i]
            $best = $a[$i];

            // if not the first index
            if ($i)

                // no of wins is 1 now
```

```
                $times = 1;
        }

        else

            // if it wins
            $times += 1;

        // if any position has more then
        // k wins then return
        if ($times >= $k)
            return $best;
    }

    // Maximum element will be winner
    // because we move smaller element
    // at end and repeat the process.
    return $best;
}

// Driver Code
$a = array( 2, 1, 3, 4, 5 );
$n = sizeof($a);
$k = 2;
echo(winner($a, $n, $k));

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

Output :

```
5
```

Time complexity : O(n)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/array-element-moved-k-using-single-moves/

Chapter 4

Assign Mice to Holes

Assign Mice to Holes - GeeksforGeeks

There are N Mice and N holes are placed in a straight line. Each hole can accommodate only 1 mouse. A mouse can stay at his position, move one step right from x to x + 1, or move one step left from x to x -1. Any of these moves consumes 1 minute. Assign mice to holes so that the time when the last mouse gets inside a hole is minimized.

Examples:

```
Input : positions of mice are:
          4 -4 2
        positions of holes are:
          4 0 5
Output :  4
Assign mouse at position x = 4 to hole at
position x = 4 : Time taken is 0 minutes
Assign mouse at position x=-4 to hole at
position x = 0 : Time taken is 4 minutes
Assign mouse at position x=2 to hole at
position x = 5 : Time taken is 3 minutes
After 4 minutes all of the mice are in the holes.
Since, there is no combination possible where
the last mouse's time is less than 4,
answer = 4.

Input :  positions of mice are:
        -10, -79, -79, 67, 93, -85, -28, -94
          positions of holes are:
         -2, 9, 69, 25, -31, 23, 50, 78
Output : 102
```

This problem can be solved using greedy strategy. We can put every mouse to its nearest

hole to minimize the time. This can be done by sorting the positions of mice and holes. This allows us to put the ith mice to the corresponding hole in the holes list. We can then find the maximum difference between the mice and corresponding hole position.
In example 2, on sorting both the lists, we find that the mouse at position -79 is the last to travel to hole 23 taking time 102.

```
sort mice positions (in any order)
sort hole positions

Loop i = 1 to N:
    update ans according to the value
    of |mice(i) - hole(i)|. It should
    be maximum of all differences.
```

Proof of correctness:
Let i1 < i2 be the positions of two mice and let j1 < j2 be the positions of two holes. It suffices to show via case analysis that

```
max(|i1-j1|, |i2-j2|) <= max(|i1-j2|, |i2-j1|),
   where '|a - b|' represent absolute value of (a - b)
```

Since it follows by induction that every assignment can be transformed by a series of swaps into the sorted assignment, where none of these swaps increases the span.

```
 // Java program to find the minimum time to place
// all mice in all holes.
import java.util.* ;

public class GFG
{
    // Returns minimum time required to place mice
    // in holes.
    public int assignHole(ArrayList<Integer> mice,
                          ArrayList<Integer> holes)
    {
        if (mice.size() != holes.size())
           return -1;

        /* Sort the lists */
        Collections.sort(mice);
        Collections.sort(holes);

        int size = mice.size();

        /* finding max difference between ith mice and hole */
```

```
        int max = 0;
        for (int i=0; i<size; i++)
            if (max < Math.abs(mice.get(i)-holes.get(i)))
                max = Math.abs(mice.get(i)-holes.get(i));

        return Math.abs(max);
    }

    /* Driver Function to test other functions */
    public static void main(String[] args)
    {
        GFG gfg = new GFG();
        ArrayList<Integer> mice = new ArrayList<Integer>();
        mice.add(4);
        mice.add(-4);
        mice.add(2);
        ArrayList<Integer> holes= new ArrayList<Integer>();
        holes.add(4);
        holes.add(0);
        holes.add(5);
        System.out.println("The last mouse gets into "+
         "the hole in time: "+gfg.assignHole(mice, holes));
    }
}
```

Output:

```
The last mouse gets into the hole in time: 4
```

Source

https://www.geeksforgeeks.org/assign-mice-holes/

Chapter 5

Bin Packing Problem (Minimize number of used Bins)

Bin Packing Problem (Minimize number of used Bins) - GeeksforGeeks

Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.

Example:

```
Input:  wieght[]       = {4, 8, 1, 4, 2, 1}
        Bin Capacity c = 10
Output: 2
We need minimum 2 bins to accommodate all items
First bin contains {4, 4, 2} and second bin {8, 2}

Input:  wieght[]       = {9, 8, 2, 2, 5, 4}
        Bin Capacity c = 10
Output: 4
We need minimum 4 bins to accommodate all items.

Input:  wieght[]       = {2, 5, 4, 7, 1, 3, 8};
        Bin Capacity c = 10
Output: 3
```

Lower Bound

We can always find a lower bound on minimum number of bins required. The lower bound can be given as :

```
Min no. of bins  >=  Ceil ((Total Weight) / (Bin Capacity))
```

In the above examples, lower bound for first example is "ceil(4 + 8 + 1 + 4 + 2 + 1)/10" = 2 and lower bound in second example is "ceil(9 + 8 + 2 + 2 + 5 + 4)/10" = 3.
This problem is a NP Hard problem and finding an exact minimum number of bins takes exponential time. Following are approximate algorithms for this problem.

Applications

1. Loading of containers like trucks.
2. Placing data on multiple disks.
3. Job scheduling.
4. Packing advertisements in fixed length radio/TV station breaks.
5. Storing a large collection of music onto tapes/CD's, etc.

Online Algorithms

These algorithms are for Bin Packing problems where items arrive one at a time (in unknown order), each must be put in a bin, before considering the next item.

1. Next Fit:
When processing next item, check if it fits in the same bin as the last item. Use a new bin only if it does not.

Below is C++ implementation for this algorithm.

```
 // C++ program to find number of bins required using
// next fit algorithm.
#include <bits/stdc++.h>
using namespace std;

// Returns number of bins required using next fit
// online algorithm
int nextFit(int weight[], int n, int c)
{
   // Initialize result (Count of bins) and remaining
   // capacity in current bin.
   int res = 0, bin_rem = c;

   // Place items one by one
   for (int i=0; i<n; i++)
```

```
   {
       // If this item can't fit in current bin
       if (weight[i] > bin_rem)
       {
          res++;  // Use a new bin
          bin_rem = c - weight[i];
       }
       else
         bin_rem -= weight[i];
   }
   return res;
}

// Driver program
int main()
{
    int weight[] = {2, 5, 4, 7, 1, 3, 8};
    int c = 10;
    int n = sizeof(weight) / sizeof(weight[0]);
    cout << "Number of bins required in Next Fit : "
         << nextFit(weight, n, c);
    return 0;
}
```

Output:

```
Number of bins required in Next Fit : 4
```

Next Fit is a simple algorithm. It requires only O(n) time and O(1) extra space to process n items.

Next Fit is 2 approximate, i.e., the number of bins used by this algorithm is bounded by twice of optimal. Consider any two adjacent bins. The sum of items in these two bins must be $>$ c; otherwise, NextFit would have put all the items of second bin into the first. The same holds for all other bins. Thus, at most half the space is wasted, and so Next Fit uses at most 2M bins if M is optimal.

2. First Fit:
When processing the next item, see if it fits in the same bin as the last item. Start a new bin only if it does not.

```
 // C++ program to find number of bins required using
// First Fit algorithm.
#include <bits/stdc++.h>
using namespace std;

// Returns number of bins required using first fit
// online algorithm
```

```
int firstFit(int weight[], int n, int c)
{
    // Initialize result (Count of bins)
    int res = 0;

    // Create an array to store remaining space in bins
    // there can be at most n bins
    int bin_rem[n];

    // Place items one by one
    for (int i=0; i<n; i++)
    {
        // Find the first bin that can accommodate
        // weight[i]
        int j;
        for (j=0; j<res; j++)
        {
            if (bin_rem[j] >= weight[i])
            {
                bin_rem[j] = bin_rem[j] - weight[i];
                break;
            }
        }

        // If no bin could accommodate weight[i]
        if (j==res)
        {
            bin_rem[res] = c - weight[i];
            res++;
        }
    }
    return res;
}

// Driver program
int main()
{
    int weight[] = {2, 5, 4, 7, 1, 3, 8};
    int c = 10;
    int n = sizeof(weight) / sizeof(weight[0]);
    cout << "Number of bins required in First Fit : "
         << firstFit(weight, n, c);
    return 0;
}
```

Output:

```
Number of bins required in First Fit : 4
```

The above implementation of First Fit requires $O(n^2)$ time, but First Fit can be implemented in O(n Log n) time using Self-Balancing Binary Search Trees.

If M is the optimal number of bins, then First Fit never uses more than 1.7M bins. So First Fit is better than Next Fit in terms of upper bound on number of bins.

3. Best Fit:
The idea is to places the next item in the *tightest* spot. That is, put it in the bin so that smallest empty space is left.

```
 // C++ program to find number of bins required using
// Best fit algorithm.
#include <bits/stdc++.h>
using namespace std;

// Returns number of bins required using best fit
// online algorithm
int bestFit(int weight[], int n, int c)
{
    // Initialize result (Count of bins)
    int res = 0;

    // Create an array to store remaining space in bins
    // there can be at most n bins
    int bin_rem[n];

    // Place items one by one
    for (int i=0; i<n; i++)
    {
        // Find the best bin that ca\n accomodate
        // weight[i]
        int j;

        // Initialize minimum space left and index
        // of best bin
        int min = c+1, bi = 0;

        for (j=0; j<res; j++)
        {
            if (bin_rem[j] >= weight[i] &&
                    bin_rem[j] - weight[i] < min)
            {
                bi = j;
                min = bin_rem[j] - weight[i];
            }
        }

        // If no bin could accommodate weight[i],
        // create a new bin
```

```
        if (min==c+1)
        {
            bin_rem[res] = c - weight[i];
            res++;
        }
        else // Assign the item to best bin
            bin_rem[bi] -= weight[i];
    }
    return res;
}

// Driver program
int main()
{
    int weight[] = {2, 5, 4, 7, 1, 3, 8};
    int c = 10;
    int n = sizeof(weight) / sizeof(weight[0]);
    cout << "Number of bins required in Best Fit : "
         << bestFit(weight, n, c);
    return 0;
}
```

Output:

```
Number of bins required in Best Fit : 4
```

Best Fit can also be implemented in O(n Log n) time using Self-Balancing Binary Search Trees.

If M is the optimal number of bins, then Best Fit never uses more than 1.7M bins. So Best Fit is same as First Fit and better than Next Fit in terms of upper bound on number of bins.

Offline Algorithms

In the offline version, we have all items upfront. Unfortunately offline version is also NP Complete, but we have a better approximate algorithm for it. First Fit Decreasing uses at most (4M + 1)/3 bins if the optimal is M.

4. First Fit Decreasing:
A trouble with online algorithms is that packing large items is difficult, especially if they occur late in the sequence. We can circumvent this by *sorting* the input sequence, and placing the large items first. With sorting, we get First Fit Decreasing and Best Fit Decreasing, as offline analogs of online First Fit and Best Fit.

```
 // C++ program to find number of bins required using
// First Fit Decreasing algorithm.
#include <bits/stdc++.h>
```

```
using namespace std;

/* Copy firstFit() from above */

// Returns number of bins required using first fit
// decreasing offline algorithm
int firstFitDec(int weight[], int n, int c)
{
    // First sort all weights in decreasing order
    sort(weight, weight+n, std::greater<int>());

    // Now call first fit for sorted items
    return firstFit(weight, n, c);
}

// Driver program
int main()
{
    int weight[] = {2, 5, 4, 7, 1, 3, 8};
    int c = 10;
    int n = sizeof(weight) / sizeof(weight[0]);
    cout << "Number of bins required in First Fit "
         << "Decreasing : " << firstFitDec(weight, n, c);
    return 0;
}
```

Output:

```
Number of bins required in First Fit Decreasing : 3
```

First Fit decreasing produces the best result for the sample input because items are sorted first.

First Fit Decreasing can also be implemented in O(n Log n) time using Self-Balancing Binary Search Trees.

Source:
https://www.cs.ucsb.edu/~suri/cs130b/BinPacking.txt

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

Source

https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/

Chapter 6

Boruvka's algorithm for Minimum Spanning Tree

Boruvka's algorithm for Minimum Spanning Tree - GeeksforGeeks

Following two algorithms are generally taught for Minimum Spanning Tree (MST) problem.

Prim's algorithm
Kruskal's algorithm

There is a third algorithm called Boruvka's algorithm for MST which (like the above two) is also Greedy algorithm. The Boruvka's algorithm is the oldest minimum spanning tree algorithm was discovered by Boruuvka in 1926, long before computers even existed. The algorithm was published as a method of constructing an efficient electricity network. See following links for the working and applications of the algorithm.

Sources:
http://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Source

https://www.geeksforgeeks.org/g-fact-83/

Chapter 7

Boruvka's algorithm Greedy Algo-9

Boruvka's algorithm Greedy Algo-9 - GeeksforGeeks

We have discussed following topics on Minimum Spanning Tree.

Applications of Minimum Spanning Tree Problem
Kruskal's Minimum Spanning Tree Algorithm
Prim's Minimum Spanning Tree Algorithm

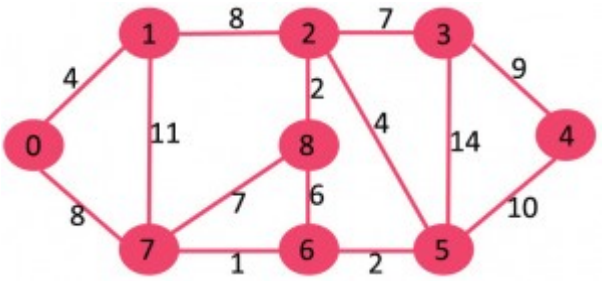
In this post, Boruvka's algorithm is discussed. Like Prim's and Kruskal's, Boruvka's algorithm is also a Greedy algorithm. Below is complete algorithm.

```
1) Input is a connected, weighted and directed graph.
2) Initialize all vertices as individual components (or sets).
3) Initialize MST as empty.
4) While there are more than one components, do following
   for each component.
      a)  Find the closest weight edge that connects this
          component to any other component.
      b)  Add this closest edge to MST if not already added.
5) Return MST.
```

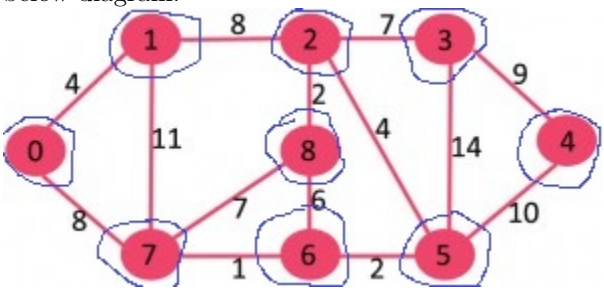
Below is the idea behind above algorithm (The idea is same as Prim's MST algorithm).

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.

Let us understand the algorithm with below example.



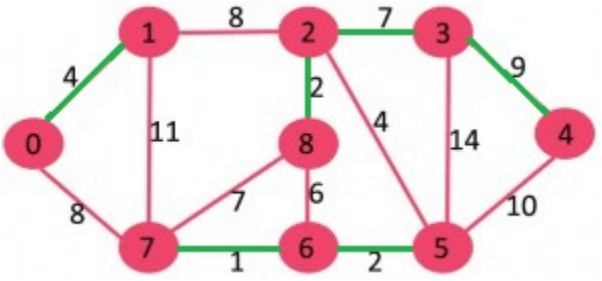
Initially MST is empty. Every vertex is singe component as highlighted in blue color in below diagram.



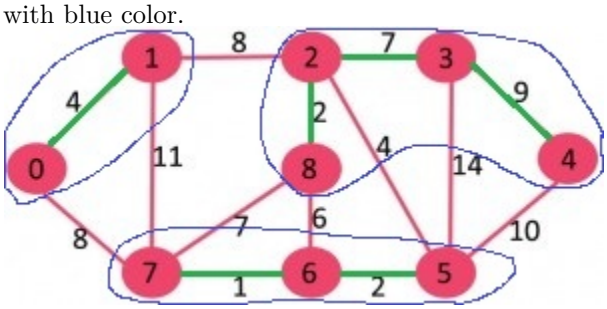
For every component, find the cheapest edge that connects it to some other component.

Component	Cheapest Edge that connects it to some other component
{0}	0-1
{1}	0-1
{2}	2-8
{3}	2-3
{4}	3-4
{5}	5-6
{6}	6-7
{7}	6-7
{8}	2-8

The cheapest edges are highlighted with green color. Now MST becomes {0-1, 2-8, 2-3, 3-4, 5-6, 6-7}.



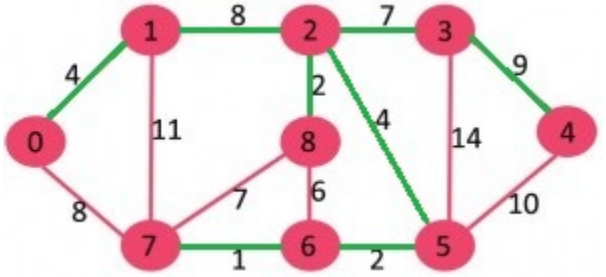
After above step, components are {{0,1}, {2,3,4,8}, {5,6,7}}. The components are encircled with blue color.



We again repeat the step, i.e., for every component, find the cheapest edge that connects it to some other component.

```
Component                Cheapest Edge that connects
                         it to some other component
  {0,1}                        1-2 (or 0-7)
  {2,3,4,8}                    2-5
  {5,6,7}                      2-5
```

The cheapest edges are highlighted with green color. Now MST becomes {0-1, 2-8, 2-3, 3-4, 5-6, 6-7, 1-2, 2-5}



At this stage, there is only one component {0, 1, 2, 3, 4, 5, 6, 7, 8} which has all edges. Since there is only one component left, we stop and return MST.

Implementation:
Below is implementation of above algorithm. The input graph is represented as a collection of edges and union-find data structure is used to keep track of components.
C/C++

```
 // Boruvka's algorithm to find Minimum Spanning
// Tree of a given connected, undirected and
// weighted graph
#include <stdio.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 as a collection of edges.
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.
    Edge* edge;
};

// A structure to represent a subset for union-find
struct subset
{
    int parent;
    int rank;
};

// Function prototypes for union-find (These functions are defined
// after boruvkaMST() )
int find(struct subset subsets[], int i);
void Union(struct subset subsets[], int x, int y);

// The main function for MST using Boruvka's algorithm
void boruvkaMST(struct Graph* graph)
{
    // Get data of given graph
    int V = graph->V, E = graph->E;
    Edge *edge = graph->edge;

    // Allocate memory for creating V subsets.
    struct subset *subsets = new subset[V];

    // An array to store index of the cheapest edge of
    // subset.  The stored index for indexing array 'edge[]'
    int *cheapest = new int[V];

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
```

```
    subsets[v].parent = v;
    subsets[v].rank = 0;
    cheapest[v] = -1;
}

// Initially there are V different trees.
// Finally there will be one tree that will be MST
int numTrees = V;
int MSTweight = 0;

// Keep combining components (or sets) until all
// compnentes are not combined into single MST.
while (numTrees > 1)
{
     // Everytime initialize cheapest array
      for (int v = 0; v < V; ++v)
       {
           cheapest[v] = -1;
       }

    // Traverse through all edges and update
    // cheapest of every component
    for (int i=0; i<E; i++)
    {
        // Find components (or sets) of two corners
        // of current edge
        int set1 = find(subsets, edge[i].src);
        int set2 = find(subsets, edge[i].dest);

        // If two corners of current edge belong to
        // same set, ignore current edge
        if (set1 == set2)
            continue;

        // Else check if current edge is closer to previous
        // cheapest edges of set1 and set2
        else
        {
           if (cheapest[set1] == -1 ||
               edge[cheapest[set1]].weight > edge[i].weight)
             cheapest[set1] = i;

           if (cheapest[set2] == -1 ||
               edge[cheapest[set2]].weight > edge[i].weight)
             cheapest[set2] = i;
        }
    }
```

```
        // Consider the above picked cheapest edges and add them
        // to MST
        for (int i=0; i<V; i++)
        {
            // Check if cheapest for current set exists
            if (cheapest[i] != -1)
            {
                int set1 = find(subsets, edge[cheapest[i]].src);
                int set2 = find(subsets, edge[cheapest[i]].dest);

                if (set1 == set2)
                    continue;
                MSTweight += edge[cheapest[i]].weight;
                printf("Edge %d-%d included in MST\n",
                       edge[cheapest[i]].src, edge[cheapest[i]].dest);

                // Do a union of set1 and set2 and decrease number
                // of trees
                Union(subsets, set1, set2);
                numTrees--;
            }
        }
    }

    printf("Weight of MST is %d\n", MSTweight);
    return;
}

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}

// 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++;
    }
}

// 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;

    boruvkaMST(graph);

    return 0;
}

// Thanks to Anukul Chand for modifying above code.
```

Python

```
 # Boruvka'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 boruvkaMST(self):
        parent = []; rank = [];

        # An array to store index of the cheapest edge of
        # subset. It store [u,v,w] for each component
        cheapest =[]

        # Initially there are V different trees.
        # Finally there will be one tree that will be MST
        numTrees = self.V
        MSTweight = 0

        # Create V subsets with single elements
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
            cheapest =[-1] * self.V

        # Keep combining components (or sets) until all
        # compnentes are not combined into single MST

        while numTrees > 1:

            # Traverse through all edges and update
               # cheapest of every component
```

```
            for i in range(len(self.graph)):

                # Find components (or sets) of two corners
                # of current edge
                u,v,w =  self.graph[i]
                set1 = self.find(parent, u)
                set2 = self.find(parent ,v)

                # If two corners of current edge belong to
                # same set, ignore current edge. Else check if
                # current edge is closer to previous
                # cheapest edges of set1 and set2
                if set1 != set2:

                    if cheapest[set1] == -1 or cheapest[set1][2] > w :
                        cheapest[set1] = [u,v,w]

                    if cheapest[set2] == -1 or cheapest[set2][2] > w :
                        cheapest[set2] = [u,v,w]

            # Consider the above picked cheapest edges and add them
            # to MST
            for node in range(self.V):

                #Check if cheapest for current set exists
                if cheapest[node] != -1:
                    u,v,w = cheapest[node]
                    set1 = self.find(parent, u)
                    set2 = self.find(parent ,v)

                    if set1 != set2 :
                        MSTweight += w
                        self.union(parent, rank, set1, set2)
                        print ("Edge %d-%d with weight %d included in MST" % (u,v,w))
                        numTrees = numTrees - 1

            #reset cheapest array
            cheapest =[-1] * self.V


        print ("Weight of MST is %d" % MSTweight)



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.boruvkaMST()

#This code is contributed by Neelam Yadav
```

Output:

```
Edge 0-3 included in MST
Edge 0-1 included in MST
Edge 2-3 included in MST
Weight of MST is 19
```

Interesting Facts about Boruvka's algorithm:
1) Time Complexity of Boruvka's algorithm is O(E log V) which is same as Kruskal's and Prim's algorithms.

2) Boruvka's algorithm is used as a step in a faster randomized algorithm that works in linear time O(E).

3) Boruvka's algorithm is the oldest minimum spanning tree algorithm was discovered by Boruuvka in 1926, long before computers even existed. The algorithm was published as a method of constructing an efficient electricity network.

Exercise:
The above code assumes that input graph is connected and it fails if a disconnected graph is given. Extend the above algorithm so that it works for a disconnected graph also and produces a forest.

References:
http://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Improved By : Cyberfreak

Source

https://www.geeksforgeeks.org/boruvkas-algorithm-greedy-algo-9/

Chapter 8

Buy Maximum Stocks if i stocks can be bought on i-th day

Buy Maximum Stocks if i stocks can be bought on i-th day - GeeksforGeeks

In a stock market, there is a product with its infinite stocks. The stock prices are given for **N** days, where arr[i] denotes the price of the stock on the i^{th} day. There is a rule that a customer can buy at most i stock on the i^{th} day. If the customer has an amount of **k** amount of money initially, find out the maximum number of stocks a customer can buy.
For example, for 3 days the price of a stock is given as 7, 10, 4. You can buy 1 stock worth 7 rs on day 1, 2 stocks worth 10 rs each on day 2 and 3 stock worth 4 rs each on day 3.

Examples:

```
Input : price[] = { 10, 7, 19 },
              k = 45.
Output : 4
A customer purchases 1 stock on day 1,
2 stocks on day 2 and 1 stock on day 3 for
10, 7 * 2 = 14 and 19 respectively. Hence,
total amount is 10 + 14 + 19 = 43 and number
of stocks purchased is 4.

Input  : price[] = { 7, 10, 4 },
               k = 100.
Output : 6
```

The idea is to use greedy approach, where we should start buying product from the day when the stock price is least and so on.
So, we will sort the pair of two values i.e { stock price, day } according to the stock price, and if stock prices are same, then we sort according to the day.
Now, we will traverse along the sorted list of pairs, and start buying as follows:

Let say, we have R rs remaining till now, and the cost of product on this day be C, and we can buy atmost L products on this day then,
total purchase on this day (P) = min(L, R/C)
Now, add this value to the answer.
total purchase on this day (P) = min(L, R/C)
Now, add this value to the answer
total_purchase = total_purchase + P, where P =min(L, R/C)
Now, substract the cost of buying P items from remaining money, R = R – P*C.
Total number of products that we can buy is equal to total_purchase.

Below is C++ implementation of this approach:

```
 // CPP program to find maximum number of stocks that
// can be bought with given constraints.
#include <bits/stdc++.h>
using namespace std;

// Return the maximum stocks
int buyMaximumProducts(int n, int k, int price[])
{
    vector<pair<int, int> > v;

    // Making pair of product cost and number
    // of day..
    for (int i = 0; i < n; ++i)
        v.push_back(make_pair(price[i], i + 1));

    // Sorting the vector pair.
    sort(v.begin(), v.end());

    // Calculating the maximum number of stock
    // count.
    int ans = 0;
    for (int i = 0; i < n; ++i) {
        ans += min(v[i].second, k / v[i].first);
        k -= v[i].first * min(v[i].second,
                              (k / v[i].first));
    }

    return ans;
}

// Driven Program
int main()
{
    int price[] = { 10, 7, 19 };
    int n = sizeof(price)/sizeof(price[0]);
    int k = 45;
```

```
    cout << buyMaximumProducts(n, k, price) << endl;

    return 0;
}
```

Output:

```
4
```

Time Complexity :O(nlogn).

Source

https://www.geeksforgeeks.org/buy-maximum-stocks-stocks-can-bought-th-day/

Chapter 9

Check if it is possible to survive on Island

Check if it is possible to survive on Island - GeeksforGeeks

You are a poor person in an island. There is only one shop in this island, this shop is open on all days of the week except for Sunday. Consider following constraints:

- N – Maximum unit of food you can buy each day.
- S – Number of days you are required to survive.
- M – Unit of food required each day to survive.

Currently, it's Monday, and you need to survive for the next S days.
Find the minimum number of days on which you need to buy food from the shop so that you can survive the next S days, or determine that it isn't possible to survive.

Examples:

> Input : S = 10 N = 16 M = 2
> Output : Yes 2
> **Explanation 1:** One possible solution is to buy a box on the first day (Monday), it's sufficient to eat from this box up to 8th day (Monday) inclusive. Now, on the 9th day (Tuesday), you buy another box and use the chocolates in it to survive the 9th and 10th day.
>
> Input : 10 20 30
> Output : No
> Explanation 2: You can't survive even if you buy food because the maximum number of units you can buy in one day is less the required food for one day.

Approach:
In this problem, the greedy approach of buying the food for some consecutive early days is the right direction.
If we can survive for the first 7 days then we can survive any number of days for that we need to check two things
-> Check whether we can survive one day or not.
-> (S >= 7) If we buy food in the first 6 days of the week and we can survive for the week i.e. total food we can buy in a week (6*N) is greater then or equal to total food we require to survive in a week (7*M) then we can survive.
If any of the above conditions are not true then we can't survive else the minimum number of days required to buy food will be = ceil(total food required/units of food we can buy each day)

CPP

```
 // C++ program to find the minimum days on which
// you need to buy food from the shop so that you
// can survive the next S days
#include <bits/stdc++.h>
using namespace std;

// function to find the minimum days
void survival(int S, int N, int M)
{

    // If we can not buy at least a week
    // supply of food during the first week
    // OR We can not buy a day supply of food
    // on the first day then we can't survive.
    if (((N * 6) < (M * 7) && S > 6) || M > N)
        cout << "No\n";
    else {
        // If we can survive then we can
        // buy ceil(A/N) times where A is
        // total units of food required.
        int days = (M * S) / N;
        if (((M * S) % N) != 0)
            days++;
        cout << "Yes " << days << endl;
    }
}

// Driver code
int main()
{
    int S = 10, N = 16, M = 2;
    survival(S, N, M);
    return 0;
}
```

Java

```
 // Java program to find the minimum days on which
// you need to buy food from the shop so that you
// can survive the next S days
import java.io.*;

class GFG {

    // function to find the minimum days
    static void survival(int S, int N, int M)
    {

        // If we can not buy at least a week
        // supply of food during the first
        // week OR We can not buy a day supply
        // of food on the first day then we
        // can't survive.
        if (((N * 6) < (M * 7) && S > 6) || M > N)
            System.out.println("No");

        else {

            // If we can survive then we can
            // buy ceil(A/N) times where A is
            // total units of food required.
            int days = (M * S) / N;

            if (((M * S) % N) != 0)
                days++;

            System.out.println("Yes " + days);
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        int S = 10, N = 16, M = 2;

        survival(S, N, M);
    }
}

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

Python3

```
 # Python3 program to find the minimum days on
# which you need to buy food from the shop so
# that you can survive the next S days
def survival(S, N, M):

# If we can not buy at least a week
# supply of food during the first week
# OR We can not buy a day supply of food
# on the first day then we can't survive.
    if (((N * 6) < (M * 7) and S > 6) or M > N):
        print("No")
    else:

    # If we can survive then we can
    # buy ceil(A / N) times where A is
    # total units of food required.
        days = (M * S) / N

        if (((M * S) % N) != 0):
            days += 1
        print("Yes "),
        print(days)

# Driver code
S = 10; N = 16; M = 2
survival(S, N, M)

# This code is contributed by upendra bartwal
```

C#

```
 // C# program to find the minimum days
// on which you need to buy food from
// the shop so that you can survive
// the next S days
using System;

class GFG {

    // function to find the minimum days
    static void survival(int S, int N, int M)
    {

        // If we can not buy at least a week
        // supply of food during the first
        // week OR We can not buy a day
        // supply of food on the first day
        // then we can't survive.
```

```
        if (((N * 6) < (M * 7) && S > 6) || M > N)
            Console.Write("No");
        else {

            // If we can survive then we can
            // buy ceil(A/N) times where A is
            // total units of food required.
            int days = (M * S) / N;

            if (((M * S) % N) != 0)
                days++;

            Console.WriteLine("Yes " + days);
        }
    }

    // Driver code
    public static void Main()
    {
        int S = 10, N = 16, M = 2;

        survival(S, N, M);
    }
}

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

PHP

```
 <?php
// PHP program to find the
// minimum days on which
// you need to buy food
// from the shop so that you
// can survive the next S days

// Function to find
// the minimum $days
function survival($S, $N, $M)
{

    // If we can not buy at least a week
    // supply of food during the first week
    // OR We can not buy a day supply of food
    // on the first day then we can't survive.
    if ((($N * 6) < ($M * 7) &&
          $S > 6) || $M >$N)
```

```
        echo "No";
    else
    {

        // If we can survive then we can
        // buy ceil(A/N) times where A is
        // total units of food required.
        $days = ($M * $S) / $N;
        if ((($M * $S) % $N) != 0)
            $days++;
        echo "Yes " , floor($days) ;
    }
}

    // Driver code
    $S = 10; $N = 16; $M = 2;
    survival($S, $N, $M);

// This code is contributed by anuj_67

?>
```

Output:

```
Yes 2
```

Improved By : Smitha Dinesh Semwal, vt_m

Source

https://www.geeksforgeeks.org/survival/

Chapter 10

Coin Change DP-7

Coin Change DP-7 - GeeksforGeeks

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 the total number of solutions, we can divide all set solutions into 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.

C++

```
 // Recursive C program for
// coin change problem.
#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;
}
```

Java

```
 // Recursive java program for
// coin change problem.
import java.io.*;

class GFG {

    // Returns the count of ways we can
    // sum S[0...m-1] coins to get sum n
    static 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
    public static void main(String[] args)
    {
        int i, j;
        int arr[] = {1, 2, 3};
        int m = arr.length;
        System.out.println( count(arr, m, 4));


    }

}

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

Python3

```
 # Recursive Python3 program for
# coin change problem.

# Returns the count of ways we can sum
# S[0...m-1] coins to get sum n
def count(S, m, 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 and 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
arr = [1, 2, 3]
m = len(arr)
print(count(arr, m, 4))

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

C#

```
 // Recursive C# program for
// coin change problem.
using System;

class GFG
{
    // Returns the count of ways we can
    // sum S[0...m-1] coins to get sum n
    static 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
    public static void Main()
    {

        int []arr = {1, 2, 3};
        int m = arr.Length;
        Console.Write( count(arr, m, 4));


    }
}
// This code is contributed by Sam007
```

PHP

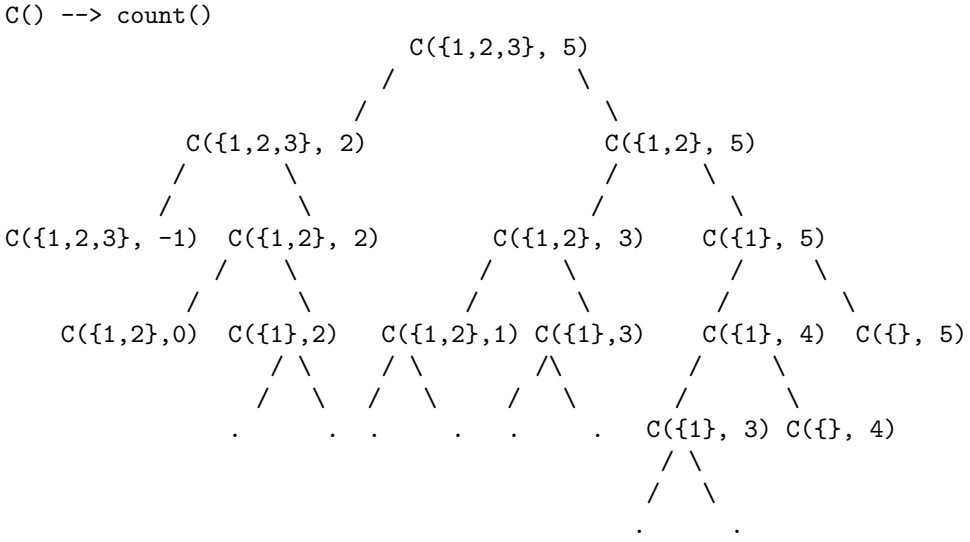
```
 <?php
// Recursive PHP program for
// coin change problem.

// Returns the count of ways we can
// sum S[0...m-1] coins to get sum n
function coun($S, $m, $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 coun($S, $m - 1,$n ) +
           coun($S, $m, $n - $S[$m - 1] );
}

    // Driver Code
    $arr = array(1, 2, 3);
    $m = count($arr);
    echo coun($arr, $m, 4);

// This code is contributed by Sam007
?>
```

Output :

```
4
```

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.



Since same suproblems are called again, this problem has Overlapping Subprolems property. So the Coin Change problem has both properties (see thisand 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.

Dynamic Programming Solution
C

```
 // C program for coin change problem.
#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 enteries 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;

            // 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;
}
```

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
```

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
```

C#

```
 /* Dynamic Programming C# implementation of Coin
Change problem */
using System;

class GFG
{
    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)
        int[] table = new int[n+1];

        // Initialize all table values as 0
        for(int i = 0; i < table.Length; i++)
        {
            table[i] = 0;
        }

        // 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
    public static void Main()
    {
        int []arr = {1, 2, 3};
        int m = arr.Length;
        int n = 4;
        Console.Write(countWays(arr, m, n));
    }
}
// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program for
// coin change problem.

function count1($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)
    $table;
    for ($i = 0; $i < $n + 1; $i++)
    for ($j = 0; $j < $m; $j++)
        $table[$i][$j] = 0;

    // Fill the enteries 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;

            // 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 Code
$arr = array(1, 2, 3);
$m = count($arr);
$n = 4;
echo count1($arr, $m, $n);

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

Output:

```
4
```

Time Complexity: O(mn)

Following is a simplified version of method 2. The auxiliary space required here is O(n) only.

C

```
 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];
}
```

Java

```
 public static 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[]=new int[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(int i=0; i<m; i++)
        for(int j=S[i]; j<=n; j++)
            table[j] += table[j-S[i]];

    return table[n];
}
```

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
```

Thanks to Rohan Laishram for suggesting this space optimized version.

References:
http://www.algorithmist.com/index.php/Coin_Change

Improved By : Sam007, khyatigrover, Mithun Kumar

Source

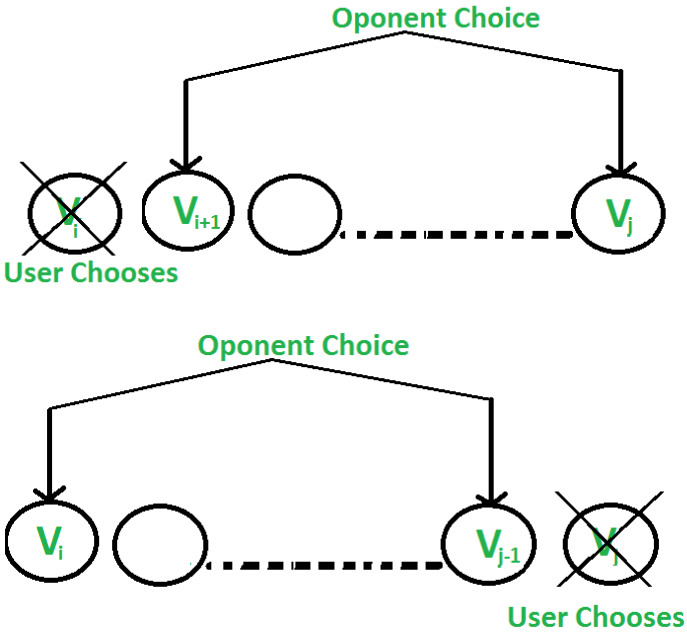
https://www.geeksforgeeks.org/coin-change-dp-7/

Chapter 11

Coin game of two corners (Greedy Approach)

Coin game of two corners (Greedy Approach) - GeeksforGeeks

Consider a two player coin game where each player gets turn one by one. There is a row of even number of coins, and a player on his/her turn can pick a coin from any of the two corners of the row. The player that collects coins with more value wins the game. Develop a strategy for the player making the first turn, such he/she never looses the game.



Note that the strategy to pick maximum of two corners may not work. In the following

example, first player looses the game when he/she uses strategy to pick maximum of two corners.

Example:

```
  18 20 15 30 10 14
First Player picks 18, now row of coins is
  20 15 30 10 14
Second player picks 20, now row of coins is
  15 30 10 14
First Player picks 15, now row of coins is
  30 10 14
Second player picks 30, now row of coins is
  10 14
First Player picks 14, now row of coins is
  10
Second player picks 10, game over.

The total value collected by second player is more (20 +
30 + 10) compared to first player (18 + 15 + 14).
So the second player wins.
```

Note that this problem is different from Optimal Strategy for a Game DP-31. There the target is to get maximum value. Here the target is to not loose. We have a Greedy Strategy here. The idea is to count sum of values of all even coins and odd coins, compare the two values. The player that makes the first move can always make sure that the other player is never able to choose an even coin if sum of even coins is higher. Similarly, he/she can make sure that the other player is never able to choose an odd coin if sum of odd coins is higher.

Example:

```
  18 20 15 30 10 14
Sum of odd coins = 18 + 15 + 10 = 43
Sum of even coins = 20 + 30 + 14 = 64.
Since the sum of even coins is more, the first
player decides to collect all even coins. He first
picks 14, now the other player can only pick a coin
(10 or 18). Whichever is picked the other player,
the first player again gets an opportunity to pick
an even coin and block all even coins.
```

C++

```
 // CPP program to find coins to be picked to make sure
// that we never loose.
#include <iostream>
```

```
using namespace std;

// Returns optimal value possible that a player can collect
// from an array of coins of size n. Note than n must be even
void printCoins(int arr[], int n)
{
    // Find sum of odd positioned coins
    int oddSum = 0;
    for (int i = 0; i < n; i += 2)
        oddSum += arr[i];

    // Find sum of even positioned coins
    int evenSum = 0;
    for (int i = 0; i < n; i += 2)
        evenSum += arr[i];

    // Print even or odd coins depending upon
    // which sum is greater.
    int start = ((oddSum > evenSum) ? 0 : 1);
    for (int i = start; i < n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above function
int main()
{
    int arr1[] = { 8, 15, 3, 7 };
    int n = sizeof(arr1) / sizeof(arr1[0]);
    printCoins(arr1, n);
    cout << endl;

    int arr2[] = { 2, 2, 2, 2 };
    n = sizeof(arr2) / sizeof(arr2[0]);
    printCoins(arr2, n);
    cout << endl;

    int arr3[] = { 20, 30, 2, 2, 2, 10 };
    n = sizeof(arr3) / sizeof(arr3[0]);
    printCoins(arr3, n);

    return 0;
}
```

Java

```
// Java program to find coins to be
// picked to make sure that we never loose.
class GFG
{
```

// Returns optimal value possible
// that a player can collect from
// an array of coins of size n.
// Note than n must be even
static void printCoins(int arr[], int n)
{
// Find sum of odd positioned coins
int oddSum = 0;
for (int i = 0; i < n; i += 2) oddSum += arr[i]; // Find sum of even positioned coins int evenSum = 0; for (int i = 0; i < n; i += 2) evenSum += arr[i]; // Print even or odd coins depending // upon which sum is greater. int start = ((oddSum > evenSum) ? 0 : 1);
for (int i = start; i < n; i++) System.out.print(arr[i]+" "); } // Driver Code public static void main(String[] args) { int arr1[] = { 8, 15, 3, 7 }; int n = arr1.length; printCoins(arr1, n); System.out.println(); int arr2[] = { 2, 2, 2, 2 }; n = arr2.length; printCoins(arr2, n); System.out.println(); int arr3[] = { 20, 30, 2, 2, 2, 10 }; n = arr3.length; printCoins(arr3, n); } } // This code is contributed by ChitraNayal [tabby title = "Python 3"]

```
 # Python 3 program to find coins
# to be picked to make sure that
# we never loose

# Returns optimal value possible
# that a player can collect from
# an array of coins of size n.
# Note than n must be even
def printCoins(arr, n) :

    oddSum = 0

    # Find sum of odd positioned coins
    for i in range(0, n, 2) :
        oddSum += arr[i]

    evenSum = 0

    # Find sum of even
    # positioned coins
    for i in range(0, n, 2) :
        evenSum += arr[i]

    # Print even or odd
    # coins depending upon
    # which sum is greater.
    if oddSum > evenSum :
        start = 0
    else :
        start = 1
```

```
    for i in range(start, n) :
        print(arr[i], end = " ")

# Driver code
if __name__ == "__main__" :

    arr1 = [8, 15, 3, 7]
    n = len(arr1)
    printCoins(arr1, n)
    print()

    arr2 = [2, 2, 2, 2]
    n = len(arr2)
    printCoins(arr2, n)
    print()

    arr3 = [20, 30, 2, 2, 2, 10]
    n = len(arr3)
    printCoins(arr3, n)

# This code is contributed by ANKITRAI1
```

C#

```
// C# program to find coins to be
// picked to make sure that we never loose.
using System;

class GFG
{

// Returns optimal value possible
// that a player can collect from
// an array of coins of size n.
// Note than n must be even
static void printCoins(int[] arr, int n)
{

// Find sum of odd positioned coins
int oddSum = 0;
for (int i = 0; i < n; i += 2) oddSum += arr[i]; // Find sum of even positioned coins int
evenSum = 0; for (int i = 0; i < n; i += 2) evenSum += arr[i]; // Print even or odd coins
depending // upon which sum is greater. int start = ((oddSum > evenSum) ? 0 : 1);
for (int i = start; i < n; i++) Console.Write(arr[i]+" "); } // Driver Code public static
void Main() { int[] arr1 = { 8, 15, 3, 7 }; int n = arr1.Length; printCoins(arr1, n); Con-
sole.Write("\n"); int[] arr2 = { 2, 2, 2, 2 }; n = arr2.Length; printCoins(arr2, n); Con-
sole.Write("\n"); int[] arr3 = { 20, 30, 2, 2, 2, 10 }; n = arr3.Length; printCoins(arr3, n); }
} // This code is contributed by ChitraNayal [tabby title="PHP"] $evenSum) ? 0 : 1);
for ($i = $start; $i < $n; $i++) echo $arr[$i]." "; } // Driver Code $arr1 = array( 8, 15, 3,
7 ); $n = sizeof($arr1); printCoins($arr1, $n); echo "\n"; $arr2 = array( 2, 2, 2, 2 ); $n =
```

sizeof($arr2); printCoins($arr2, $n); echo "\n"; $arr3 = array(20, 30, 2, 2, 2, 10); $n = sizeof($arr3); printCoins($arr3, $n); // This code is contributed by ChitraNayal ?>

Output:

```
15 3 7
2 2 2
30 2 2 2 10
```

Improved By : ANKITRAI1, ChitraNayal

Source

https://www.geeksforgeeks.org/coin-game-of-two-corners-greedy-approach/

Chapter 12

Connect n ropes with minimum cost

Connect n ropes with minimum cost - GeeksforGeeks

There are given n ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. We need to connect the ropes with minimum cost.

For example if we are given 4 ropes of lengths 4, 3, 2 and 6. We can connect the ropes in following ways.
1) First connect ropes of lengths 2 and 3. Now we have three ropes of lengths 4, 6 and 5.
2) Now connect ropes of lengths 4 and 5. Now we have two ropes of lengths 6 and 9.
3) Finally connect the two ropes and all ropes have connected.

Total cost for connecting all ropes is 5 + 9 + 15 = 29. This is the optimized cost for connecting ropes. Other ways of connecting ropes would always have same or more cost. For example, if we connect 4 and 6 first (we get three strings of 3, 2 and 10), then connect 10 and 3 (we get two strings of 13 and 2). Finally we connect 13 and 2. Total cost in this way is 10 + 13 + 15 = 38.

If we observe the above problem closely, we can notice that the lengths of the ropes which are picked first are included more than once in total cost. Therefore, the idea is to connect smallest two ropes first and recur for remaining ropes. This approach is similar to Huffman Coding. We put smallest ropes down the tree so that they can be repeated multiple times rather than the longer ropes.

Following is complete algorithm for finding the minimum cost for connecting n ropes.
Let there be n ropes of lengths stored in an array len[0..n-1]
1) Create a min heap and insert all lengths into the min heap.
2) Do following while number of elements in min heap is not one.
……a) Extract the minimum and second minimum from min heap
……b) Add the above two extracted values and insert the added value to the min-heap.
……c) Maintain a variable for total cost and keep incrementing it by the sum of extracted

values.
3) Return the value of this total cost.

Following is C++ implementation of above algorithm.

```
 // C++ program for connecting n ropes with minimum cost
#include <iostream>

using namespace std;

// A Min Heap:  Collection of min heap nodes
struct MinHeap
{
    unsigned size;    // Current size of min heap
    unsigned capacity;   // capacity of min heap
    int *harr;  // Attay of minheap nodes
};

// A utility function to create a min heap of given capacity
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap = new MinHeap;
    minHeap->size = 0;  // current size is 0
    minHeap->capacity = capacity;
    minHeap->harr = new int[capacity];
    return minHeap;
}

// A utility function to swap two min heap nodes
void swapMinHeapNode(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// 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->harr[left] < minHeap->harr[smallest])
      smallest = left;

    if (right < minHeap->size &&
        minHeap->harr[right] < minHeap->harr[smallest])
```

```
      smallest = right;

    if (smallest != idx)
    {
        swapMinHeapNode(&minHeap->harr[smallest], &minHeap->harr[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
int extractMin(struct MinHeap* minHeap)
{
    int temp = minHeap->harr[0];
    minHeap->harr[0] = minHeap->harr[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, int val)
{
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && (val < minHeap->harr[(i - 1)/2]))
    {
        minHeap->harr[i] = minHeap->harr[(i - 1)/2];
        i = (i - 1)/2;
    }
    minHeap->harr[i] = val;
}

// A standard funvtion 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);
}

// Creates a min heap of capacity equal to size and inserts all values
```

```
// from len[] in it. Initially size of min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(int len[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->harr[i] = len[i];
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

// The main function that returns the minimum cost to connect n ropes of
// lengths stored in len[0..n-1]
int minCost(int len[], int n)
{
    int cost = 0;  // Initialize result

    // Create a min heap of capacity equal to n and put all ropes in it
    struct MinHeap* minHeap = createAndBuildMinHeap(len, n);

    // Iterate while size of heap doesn't become 1
    while (!isSizeOne(minHeap))
    {
        // Extract two minimum length ropes from min heap
        int min     = extractMin(minHeap);
        int sec_min = extractMin(minHeap);

        cost += (min + sec_min);  // Update total cost

        // Insert a new rope in min heap with length equal to sum
        // of two extracted minimum lengths
        insertMinHeap(minHeap, min+sec_min);
    }

    // Finally return total minimum cost for connecting all ropes
    return cost;
}

// Driver program to test above functions
int main()
{
    int len[] = {4, 3, 2, 6};
    int size = sizeof(len)/sizeof(len[0]);
    cout << "Total cost for connecting ropes is " << minCost(len, size);
    return 0;
}
```

Output:

```
Total cost for connecting ropes is 29
```

Time Complexity: Time complexity of the algorithm is O(nLogn) assuming that we use a O(nLogn) sorting algorithm. Note that heap operations like insert and extract take O(Logn) time.

Algorithmic Paradigm: Greedy Algorithm

A simple implementation with STL in C++
Following is a simple implementation that uses priority_queue available in STL. Thanks to Pango89 for providing below code.
C++

```
 #include<iostream>
#include<queue>

using namespace std;

int minCost(int arr[], int n)
{
    // Create a priority queue ( http://www.cplusplus.com/reference/queue/priority_queue/ )
    // By default 'less' is used which is for decreasing order
    // and 'greater' is used for increasing order
    priority_queue< int, vector<int>, greater<int> > pq(arr, arr+n);

    // Initialize result
    int res = 0;

    // While size of priority queue is more than 1
    while (pq.size() > 1)
    {
        // Extract shortest two ropes from pq
        int first = pq.top();
        pq.pop();
        int second = pq.top();
        pq.pop();

        // Connect the ropes: update result and
        // insert the new rope to pq
        res += first + second;
        pq.push(first + second);
    }

    return res;
}

// Driver program to test above function
int main()
{
```

```
    int len[] = {4, 3, 2, 6};
    int size = sizeof(len)/sizeof(len[0]);
    cout << "Total cost for connecting ropes is " << minCost(len, size);
    return 0;
}
```

Java

```
 // Java program to connect n
// ropes with minimum cost
import java.util.*;

class ConnectRopes
{
static int minCost(int arr[], int n)
{
    // Create a priority queue
    PriorityQueue<Integer> pq =
                        new PriorityQueue<Integer>();

    // Adding items to the pQueue
    for(int i=0;i<n;i++)
    {
        pq.add(arr[i]);
    }

    // Initialize result
    int res = 0;

    // While size of priority queue
    // is more than 1
    while (pq.size() > 1)
    {
        // Extract shortest two ropes from pq
        int first = pq.poll();
        int second = pq.poll();

        // Connect the ropes: update result
        // and insert the new rope to pq
        res += first + second;
        pq.add(first + second);
    }

    return res;
}

// Driver program to test above function
public static void main(String args[])
```

```
{
    int len[] = {4, 3, 2, 6};
    int size = len.length;
    System.out.println("Total cost for connecting"+
                " ropes is " + minCost(len, size));

}
}
// This code is contributed by yash_pec
```

Output:

```
Total cost for connecting ropes is 29
```

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

Improved By : JAGRITIBANSAL, AbhijeetSrivastava

Source

https://www.geeksforgeeks.org/connect-n-ropes-minimum-cost/

Chapter 13

Correctness of Greedy Algorithms

Correctness of Greedy Algorithms - GeeksforGeeks

A greedy algorithm selects a candidate greedily (local optimum) and adds it to the current solution provided that it doesn't corrupt the feasibility. If the solution obtained by above step is not final, repeat till global optimum or the final solution is obtained.

Although there are several mathematical strategies available to proof the correctness of Greedy Algorithms, we will try to proof it intuitively and use method of contradiction.

Greedy Algorithm usually involves a sequence of choices.Greedy algorithms can't back-track,hence once they make a choice, they're committed to it. So it's critical that they never make a bad choice.

Suppose **S** be the solution obtained by applying greedy algorithm to a problem and **O** be the optimum solution to the problem. If both **S** and **O** are same then our algorithm is by default correct. If **S** and **O** are different then clearly while stacking up various local solutions for the problem we made a mistake and chose a less efficient solution which resulted in **S** rather than **O** as a solution. But according to the definition of greedy algorithms we always choose the local optimum solution.
Hence using proof by contradiction it can said that greedy algorithm gives the correct solution.

The above proof can be understood better with help of Krushkal's Algorithm.

Kruskal's Algorithm:
This is a greedy algorithm used to find the minimum spanning tree of a graph.

Kruskal's algorithm can be stated as follows:
0. Create a minimum spanning tree T that initially contains no edges,
1. Choose an edge e in G, where
(a) e is not in T and …
(b) e is of minimum weight and …

(c) e does not create a cycle in T
2. If T does not contain all the vertices of G go to step 1.

Let **T** be a the tree obtained and **S** be the desired tree such that **W(T) > W(S)**.
Clearly for this to happen, at some point of time we chose an edge which is not having a minimum weight, but according to the above algorithm we always choose the minimum weight edges. Hence Krushkal's Algorithm will always give the correct result. Note that a Minimum Spanning Tree of V vertices must have at least V-1 edges and should not contain cycle. So we did not pick any extra edge in above step.

Source

https://www.geeksforgeeks.org/correctness-greedy-algorithms/

Chapter 14

Delete an element from array (Using two traversals and one traversal)

Delete an element from array (Using two traversals and one traversal) - GeeksforGeeks

Given an array and a number ‘x’, write a function to delete ‘x’ from the given array.

```
Input:  arr[] = {3, 1, 2, 5, 90}, x = 2
Output: arr[] = {3, 1, 5, 90}
```

A simple solution is to first search ‘x’ in array, then elements that are on right side of x to one position back. The following are the implementation of this simple approach.

C/C++

```
 // C++ program to remove a given element from an array
#include<iostream>
using namespace std;

// This function removes an element x from arr[] and
// returns new size after removal (size is reduced only
// when x is present in arr[]
int deleteElement(int arr[], int n, int x)
{
   // Search x in array
   int i;
   for (i=0; i<n; i++)
      if (arr[i] == x)
         break;
```

```
   // If x found in array
   if (i < n)
   {
     // reduce size of array and move all
     // elements on space ahead
     n = n - 1;
     for (int j=i; j<n; j++)
        arr[j] = arr[j+1];
   }

   return n;
}

/* Driver program to test above function */
int main()
{
    int arr[] = {11, 15, 6, 8, 9, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 6;

    // Delete x from arr[]
    n = deleteElement(arr, n, x);

    cout << "Modified array is \n";
    for (int i=0; i<n; i++)
       cout << arr[i] << " ";

    return 0;
}
```

Java

```
 // Java program to remove a given element from an array
import java.io.*;

class Deletion {

    // This function removes an element x from arr[] and
    // returns new size after removal (size is reduced only
    // when x is present in arr[]
    static int deleteElement(int arr[], int n, int x)
    {
        // Search x in array
        int i;
        for (i=0; i<n; i++)
            if (arr[i] == x)
                break;
```

```
        // If x found in array
        if (i < n)
        {
            // reduce size of array and move all
            // elements on space ahead
            n = n - 1;
            for (int j=i; j<n; j++)
                arr[j] = arr[j+1];
        }

        return n;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int arr[] = {11, 15, 6, 8, 9, 10};
        int n = arr.length;
        int x = 6;

        // Delete x from arr[]
        n = deleteElement(arr, n, x);

        System.out.println("Modified array is");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i]+" ");

    }
}
/*This code is contributed by Devesh Agrawal*/
```

C#

```
 // C# program to remove a given element from
// an array
using System;

class GFG {

    // This function removes an element x
    // from arr[] and returns new size
    // after removal (size is reduced only
    // when x is present in arr[]
    static int deleteElement(int []arr,
                             int n, int x)
    {
```

```
        // Search x in array
        int i;
        for (i = 0; i < n; i++)
            if (arr[i] == x)
                break;

        // If x found in array
        if (i < n)
        {

            // reduce size of array and
            // move all elements on
            // space ahead
            n = n - 1;
            for (int j = i; j < n; j++)
                arr[j] = arr[j+1];
        }

        return n;
    }

    // Driver program to test above function
    public static void Main()
    {
        int []arr = {11, 15, 6, 8, 9, 10};
        int n = arr.Length;
        int x = 6;

        // Delete x from arr[]
        n = deleteElement(arr, n, x);

        Console.WriteLine("Modified array is");
        for (int i = 0; i < n; i++)
            Console.Write(arr[i]+" ");

    }
}

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

Output:

```
Modified array is
11 15 8 9 10
```

The above method requires two traversals of array, one for searching and other for moving elements.

Can we delete the element using one traversal?
This is possible under the assumption that the element is always present in array. The idea is to start from right most element and keep moving elements while searching for 'x'. Below are C++ and Java implementations of this approach. Note that this approach may give unexpected result when 'x' is not present in array.

C++

```
 // C++ program to remove a given element from an array
#include<iostream>
using namespace std;

// This function removes an element x from arr[] and
// returns new size after removal.
// Returned size is n-1 when element is present.
// Otherwise 0 is returned to indicate failure.
int deleteElement(int arr[], int n, int x)
{
   // If x is last element, nothing to do
   if (arr[n-1] == x)
       return (n-1);

   // Start from rightmost element and keep moving
   // elements one position ahead.
   int prev = arr[n-1], i;
   for (i=n-2; i>=0 && arr[i]!=x; i--)
   {
       int curr = arr[i];
       arr[i] = prev;
       prev = curr;
   }

   // If element was not found
   if (i < 0)
     return 0;

   // Else move the next element in place of x
   arr[i] = prev;

   return (n-1);
}

/* Driver program to test above function */
int main()
{
    int arr[] = {11, 15, 6, 8, 9, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 6;
```

```
    // Delete x from arr[]
    n = deleteElement(arr, n, x);

    cout << "Modified array is \n";
    for (int i=0; i<n; i++)
       cout << arr[i] << " ";

    return 0;
}
```

Java

```
 // Java program to remove a given element from an array
import java.io.*;

class Deletion
{
    // This function removes an element x from arr[] and
    // returns new size after removal.
    // Returned size is n-1 when element is present.
    // Otherwise 0 is returned to indicate failure.
    static int deleteElement(int arr[], int n, int x)
    {
        // If x is last element, nothing to do
        if (arr[n-1] == x)
            return (n-1);

        // Start from rightmost element and keep moving
        // elements one position ahead.
        int prev = arr[n-1], i;
        for (i=n-2; i>=0 && arr[i]!=x; i--)
        {
            int curr = arr[i];
            arr[i] = prev;
            prev = curr;
        }

        // If element was not found
        if (i < 0)
            return 0;

        // Else move the next element in place of x
        arr[i] = prev;

        return (n-1);
    }

    // Driver program to test above function
```

```
    public static void main(String[] args)
    {
        int arr[] = {11, 15, 6, 8, 9, 10};
        int n = arr.length;
        int x = 6;

        // Delete x from arr[]
        n = deleteElement(arr, n, x);

        System.out.println("Modified array is");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i]+" ");

    }
}
/*This code is contributed by Devesh Agrawal*/
```

C#

```
 // C# program to remove a given
// element from an array
using System;
class GFG {

    // This function removes an
    // element x from arr[] and
    // returns new size after
    // removal. Returned size is
    // n-1 when element is present.
    // Otherwise 0 is returned to
    // indicate failure.
    static int deleteElement(int []arr,
                             int n,
                             int x)
    {

        // If x is last element,
        // nothing to do
        if (arr[n - 1] == x)
            return (n - 1);

        // Start from rightmost
        // element and keep moving
        // elements one position ahead.
        int prev = arr[n - 1], i;
        for (i = n - 2; i >= 0 &&
                arr[i] != x; i--)
        {
```

```
            int curr = arr[i];
            arr[i] = prev;
            prev = curr;
        }

        // If element was
        // not found
        if (i < 0)
            return 0;

        // Else move the next
        // element in place of x
        arr[i] = prev;

        return (n - 1);
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {11, 15, 6, 8, 9, 10};
        int n = arr.Length;
        int x = 6;

        // Delete x from arr[]
        n = deleteElement(arr, n, x);

        Console.WriteLine("Modified array is");
        for(int i = 0; i < n; i++)
            Console.Write(arr[i]+" ");

    }
}

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

Output:

```
Modified array is
11 15 8 9 10
```

Deleting an element from an array takes O(n) time even if we are given index of the element to be deleted. The time complexity remains O(n) for sorted arrays as well.
In linked list, if we know the pointer to the previous node of the node to be deleted, we can do deletion in O(1) time.

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

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/delete-an-element-from-array-using-two-traversals-and-one-traversal/

Chapter 15

Dial's Algorithm (Optimized Dijkstra for small range weights)

Dial's Algorithm (Optimized Dijkstra for small range weights) - GeeksforGeeks

Dijkstra's shortest path algorithm runs in O(Elog V) time when implemented with adjacency list representation (See C implementation and STL based C++ implementations for details).

```
Input : Source = 0, Maximum Weight W = 14
Output :
     Vertex   Distance from Source
        0                0
        1                4
        2                12
        3                19
        4                21
        5                11
        6                9
        7                8
        8                14
```

Can we optimize Dijkstra's shortest path algorithm to work better than O(E log V) if maximum weight is small (or range of edge weights is small)?
For example, in the above diagram, maximum weight is 14. Many a times the range of weights on edges in is in small range (i.e. all edge weight can be mapped to 0, 1, 2.. w where w is a small number). In that case, Dijkstra's algorithm can be modified by using different data structure, buckets, which is called dial implementation of dijkstra's algorithm.

time complexity is **O(E + WV)** where W is maximum weight on any edge of graph, so we can see that, if W is small then this implementation runs much faster than traditional algorithm. Following are important observations.

- Maximum distance between any two node can be at max w(V – 1) (w is maximum edge weight and we can have at max V-1 edges between two vertices).
- In Dijkstra algorithm, distances are finalized in non-decreasing, i.e., distance of the closer (to given source) vertices is finalized before the distant vertices.

Algorithm

Below is complete algorithm:

1. Maintains some buckets, numbered 0, 1, 2,…,wV.
2. Bucket k contains all temporarily labeled nodes with distance equal to k.
3. Nodes in each bucket are represented by list of vertices.
4. Buckets 0, 1, 2,..wV are checked sequentially until the first non-empty bucket is found. Each node contained in the first non-empty bucket has the minimum distance label by definition.
5. One by one, these nodes with minimum distance label are permanently labeled and deleted from the bucket during the scanning process.
6. Thus operations involving vertex include:
    - Checking if a bucket is empty
    - Adding a vertex to a bucket
    - Deleting a vertex from a bucket.
7. The position of a temporarily labeled vertex in the buckets is updated accordingly when the distance label of a vertex changes.
8. Process repeated until all vertices are permanently labeled (or distances of all vertices are finalized).

Implementation

Since the maximum distance can be w(V – 1), we create wV buckets (more for simplicity of code) for implementation of algorithm which can be large if w is big.

```
 // C++ Program for Dijkstra's dial implementation
#include<bits/stdc++.h>
using namespace std;
# define INF 0x3f3f3f3f

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V;  // No. of vertices

    // In a weighted graph, we need to store vertex
    // and weight pair for every edge
    list< pair<int, int> > *adj;

public:
    Graph(int V);  // Constructor

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

    // prints shortest path from s
    void shortestPath(int s, int W);
};

// Allocates memory for adjacency list
Graph::Graph(int V)
{
    this->V = V;
    adj = new list< pair<int, int> >[V];
}

//  adds edge between u and v of weight w
void Graph::addEdge(int u, int v, int w)
{
    adj[u].push_back(make_pair(v, w));
    adj[v].push_back(make_pair(u, w));
}

// Prints shortest paths from src to all other vertices.
// W is the maximum weight of an edge
void Graph::shortestPath(int src, int W)
{
    /* With each distance, iterator to that vertex in
       its bucket is stored so that vertex can be deleted
       in O(1) at time of updation. So
    dist[i].first = distance of ith vertex from src vertex
    dits[i].second = iterator to vertex i in bucket number */
```

```
vector<pair<int, list<int>::iterator> > dist(V);

// Initialize all distances as infinite (INF)
for (int i = 0; i < V; i++)
    dist[i].first = INF;

// Create buckets B[].
// B[i] keep vertex of distance label i
list<int> B[W * V + 1];

B[0].push_back(src);
dist[src].first = 0;

//
int idx = 0;
while (1)
{
    // Go sequentially through buckets till one non-empty
    // bucket is found
    while (B[idx].size() == 0 && idx < W*V)
        idx++;

    // If all buckets are empty, we are done.
    if (idx == W * V)
        break;

    // Take top vertex from bucket and pop it
    int u = B[idx].front();
    B[idx].pop_front();

    // Process all adjacents of extracted vertex 'u' and
    // update their distanced if required.
    for (auto i = adj[u].begin(); i != adj[u].end(); ++i)
    {
        int v = (*i).first;
        int weight = (*i).second;

        int du = dist[u].first;
        int dv = dist[v].first;

        // If there is shorted path to v through u.
        if (dv > du + weight)
        {
            // If dv is not INF then it must be in B[dv]
            // bucket, so erase its entry using iterator
            // in O(1)
            if (dv != INF)
                B[dv].erase(dist[v].second);
```

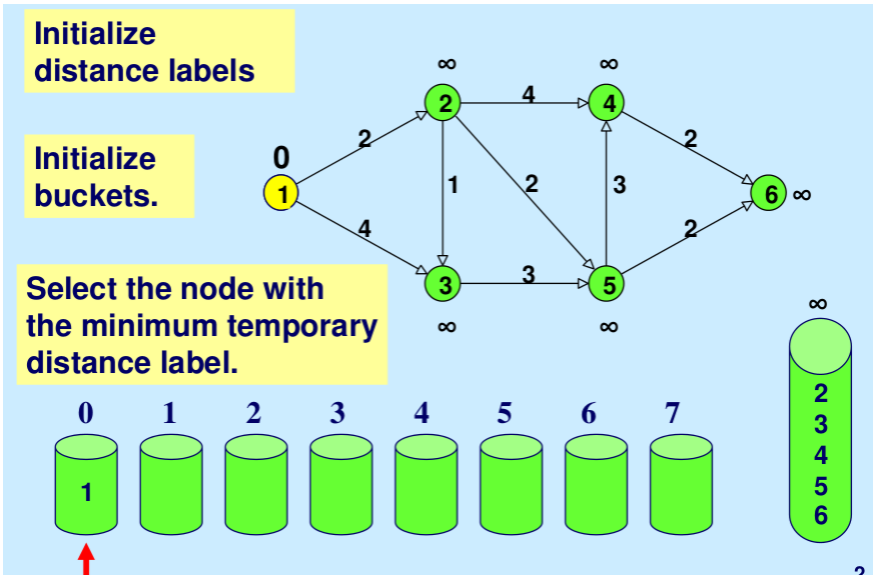
```
                //  updating the distance
                dist[v].first = du + weight;
                dv = dist[v].first;

                // pushing vertex v into updated distance's bucket
                B[dv].push_front(v);

                // storing updated iterator in dist[v].second
                dist[v].second = B[dv].begin();
            }
        }
    }

    // Print shortest distances stored in dist[]
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < V; ++i)
        printf("%d     %d\n", i, dist[i].first);
}

// Driver program to test methods of graph class
int main()
{
    // create the graph given in above fugure
    int V = 9;
    Graph g(V);

    //  making above shown graph
    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    //  maximum weighted edge - 14
    g.shortestPath(0, 14);

    return 0;
}
```

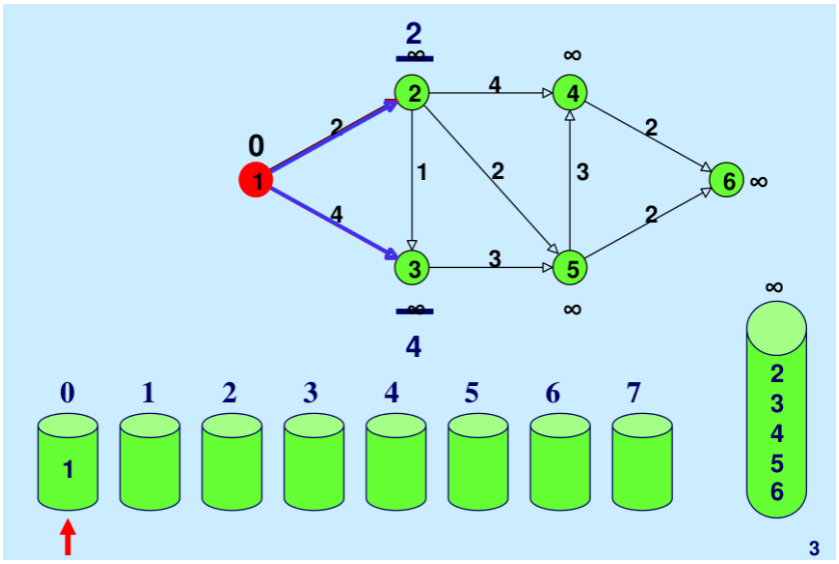
Output:

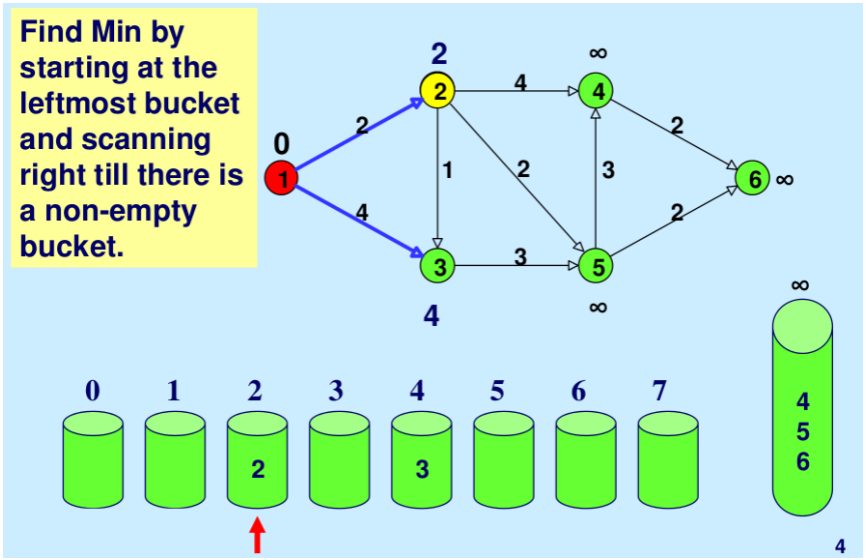
```
Vertex Distance from Source
0      0
1      4
2      12
3      19
4      21
5      11
6      9
7      8
8      14
```

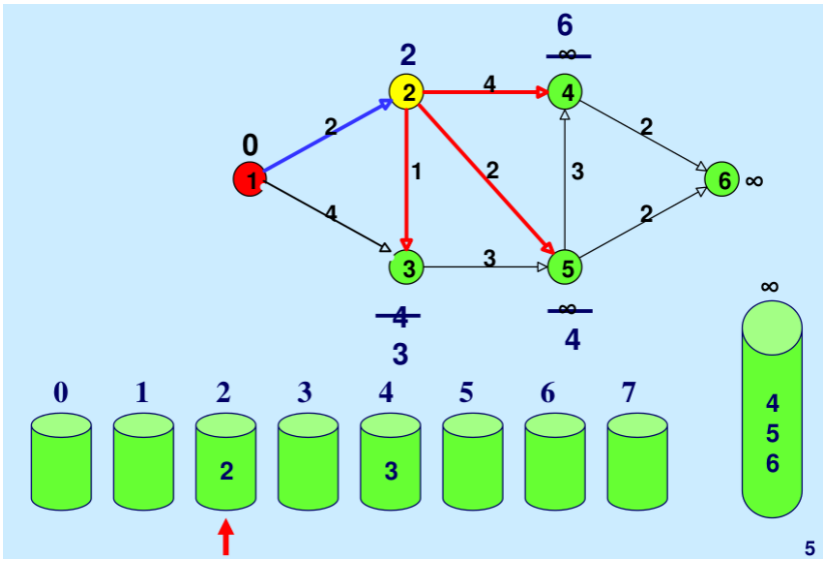
Illustration

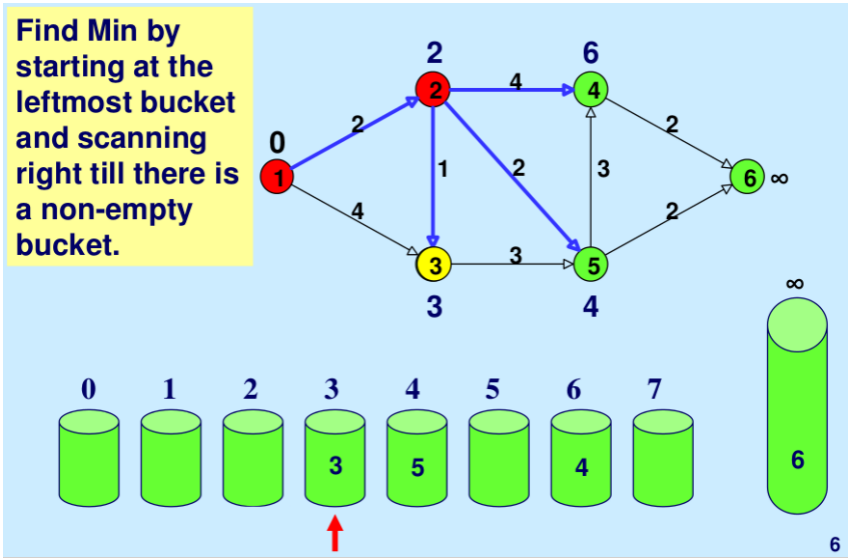
Below is step by step illustration taken from here.

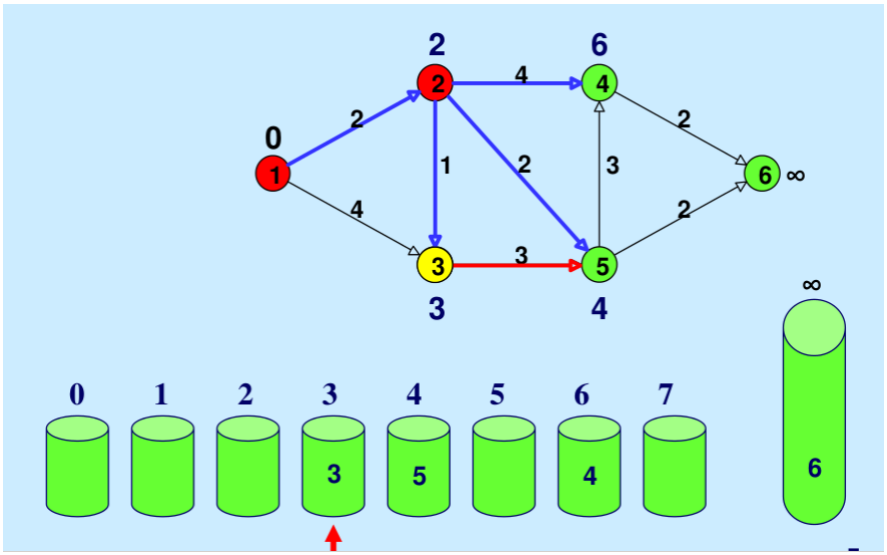


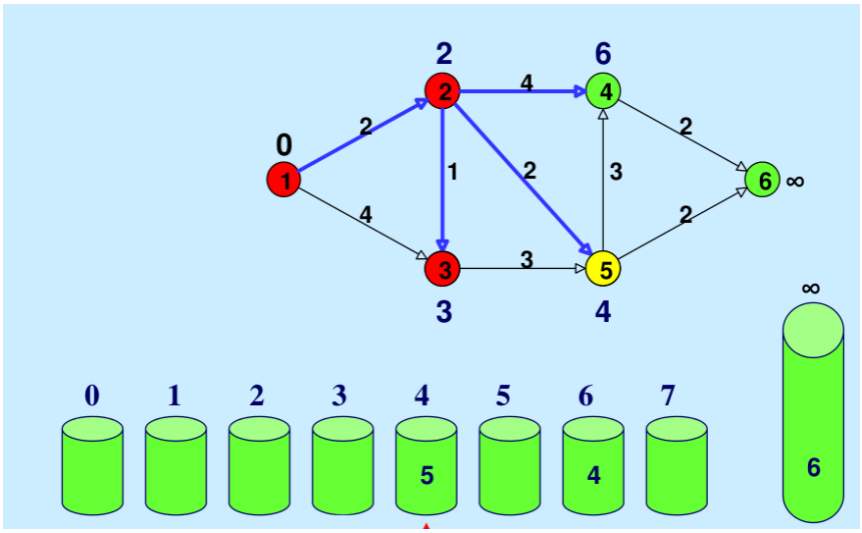

2
∞
∞
0
1
2
4
4
2
1
2
3
6
∞
4
2
3
3
5
∞
4
∞
0 1 2 3 4 5 6 7
1
2
3
4
5
6
3

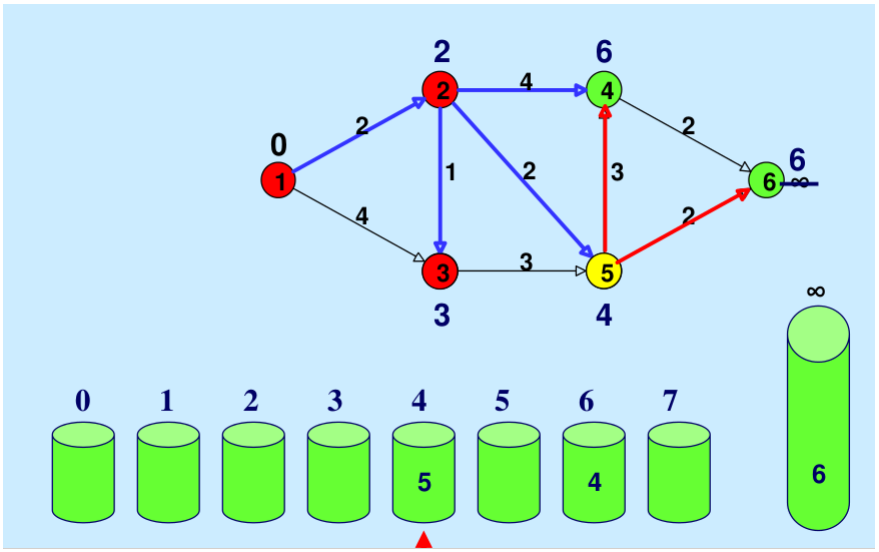

Find Min by starting at the leftmost bucket and scanning right till there is a non-empty bucket.
2
∞
∞
0
1
2
4
4
2
1
2
3
6
∞
4
2
3
3
5
∞
4
∞
0 1 2 3 4 5 6 7
2
3
4
5
6
4

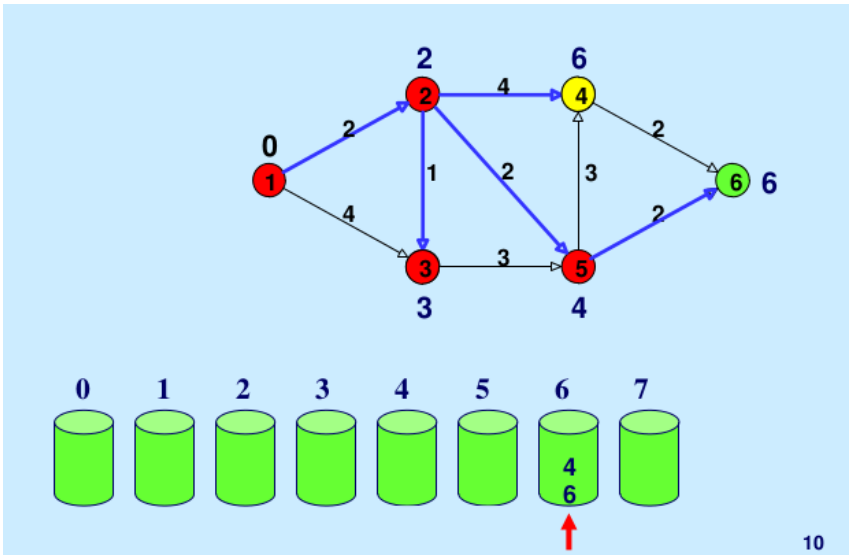

6
∞
2
2
4
4
0
1
2
1
2
3
2
6
∞
4
4
2
3
3
5
∞
4
3
0 1 2 3 4 5 6 7
2
3
∞
4
5
6
5

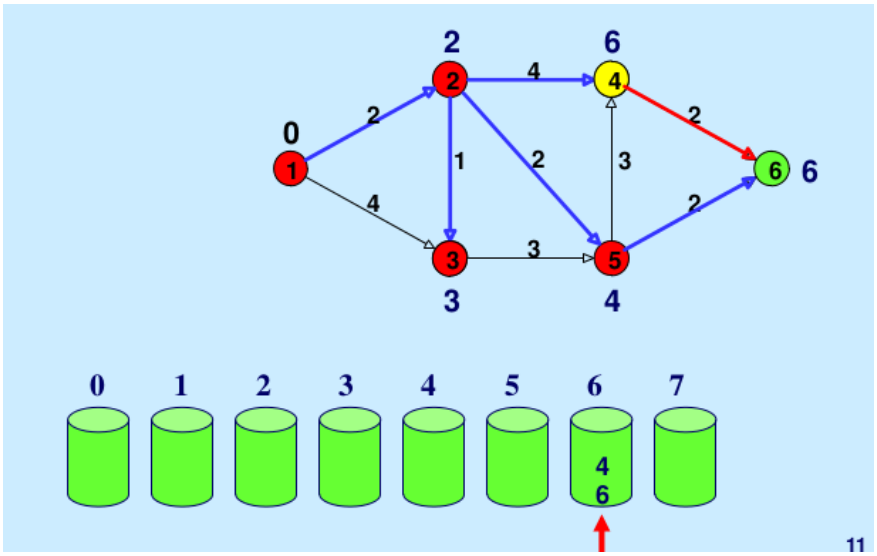

Find Min by starting at the leftmost bucket and scanning right till there is a non-empty bucket.
2
6
2
4
4
0
1
2
1
2
3
2
6
∞
4
2
3
3
5
3
4
0 1 2 3 4 5 6 7
3
5
4
∞
6
6

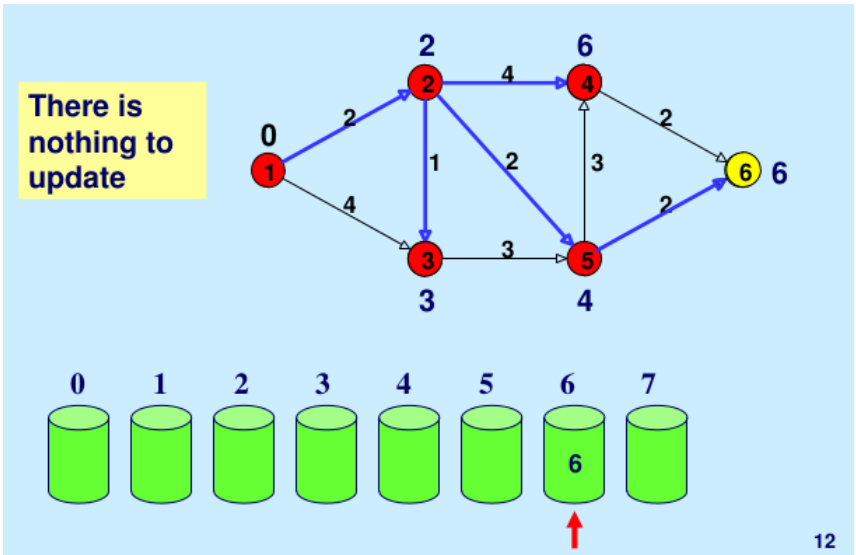

2
6
0
1
2
3
4
5
6
∞
2
4
1
2
3
2
4
3
3
4
∞
0 1 2 3 4 5 6 7
3
5
4
6

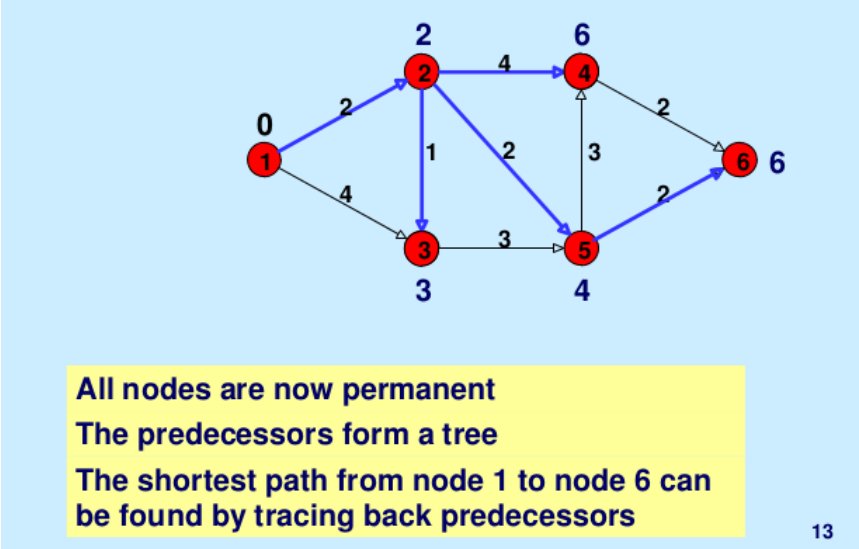

2
6
0
1
2
3
4
5
6
∞
2
4
1
2
3
2
4
3
3
4
∞
0 1 2 3 4 5 6 7
5
4
6


2
6
0
2
4
2
1
2
3
6
∞
4
2
3
3
5
3
4
∞
0 1 2 3 4 5 6 7
5
4
6


2
6
0
2
4
2
1
2
3
6
6
4
2
3
3
5
3
4
0 1 2 3 4 5 6 7
4
6
10


2
6
0
4
2
1
2
3
2
6
6
4
3
3
2
3
4
0 1 2 3 4 5 6 7
4
6
11


There is nothing to update
2
6
0
4
2
1
2
3
2
6
6
4
3
3
2
3
4
0 1 2 3 4 5 6 7
6
12



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.

Source

https://www.geeksforgeeks.org/dials-algorithm-optimized-dijkstra-for-small-range-weights/

Chapter 16

Dijkstra's Algorithm for Adjacency List Representation Greedy Algo-8

Dijkstra's Algorithm for Adjacency List Representation Greedy Algo-8 - GeeksforGeeks

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

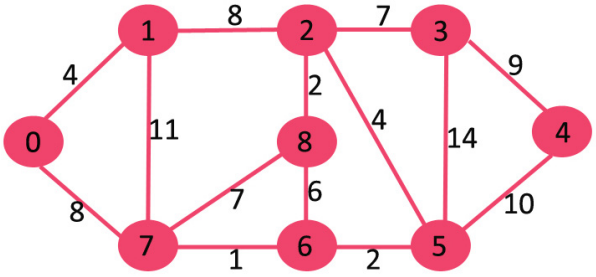
We have discussed Dijkstra's algorithm and its implementation for adjacency matrix representation of graphs. The time complexity for the matrix representation is O(V^2). In this post, O(ELogV) algorithm for adjacency list representation is discussed.

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(LogV) for Min Heap.

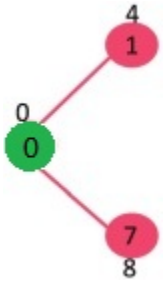
Following are the detailed steps.
1) Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and distance value of the vertex.
2) Initialize Min Heap with source vertex as root (the distance value assigned to source vertex is 0). 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.

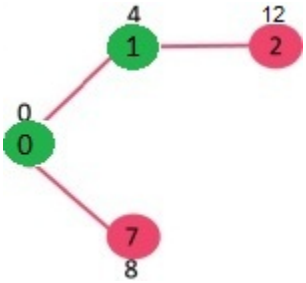
Let us understand with the following example. Let the given source vertex be 0



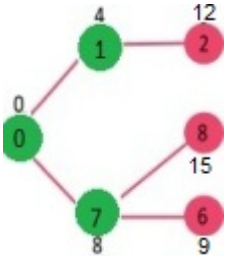
Initially, distance value of source vertex is 0 and INF (infinite) for all other vertices. So source vertex is extracted from Min Heap and distance values of vertices adjacent to 0 (1 and 7) are updated. Min Heap contains all vertices except vertex 0.
The vertices in green color are the vertices for which minimum distances are finalized and are not in Min Heap



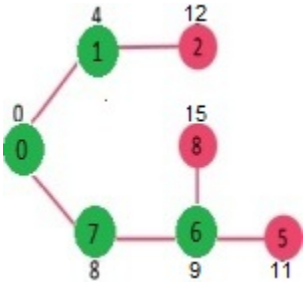
Since distance value of vertex 1 is minimum among all nodes in Min Heap, it is extracted from Min Heap and distance values of vertices adjacent to 1 are updated (distance is updated if the a vertex is not in Min Heap and distance through 1 is shorter than the previous distance). Min Heap contains all vertices except vertex 0 and 1.



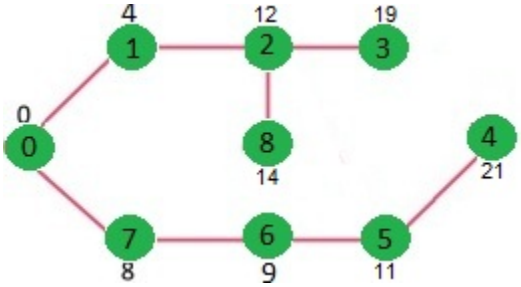
Pick the vertex with minimum distance value from min heap. Vertex 7 is picked. So min heap now contains all vertices except 0, 1 and 7. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



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 / 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 liat
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 calulates 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 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);

    dijkstra(graph, 0);

    return 0;
}
```

Python

```
 # A Python program for Dijkstra's shortest
# path algorithm for adjacency
# list representation of graph

from collections import defaultdict
import sys

class Heap():

    def __init__(self):
        self.array = []
        self.size = 0
        self.pos = []

    def newMinHeapNode(self, v, dist):
        minHeapNode = [v, dist]
        return minHeapNode

    # A utility function to swap two nodes
    # of min heap. Needed for min heapify
    def swapMinHeapNode(self,a, b):
        t = self.array[a]
        self.array[a] = self.array[b]
        self.array[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()
    def minHeapify(self, idx):
        smallest = idx
        left = 2*idx + 1
        right = 2*idx + 2

        if left < self.size and self.array[left][1] \
                                < self.array[smallest][1]:
```

```
            smallest = left

        if right < self.size and self.array[right][1]\
                                < self.array[smallest][1]:
            smallest = right

        # The nodes to be swapped in min
        # heap if idx is not smallest
        if smallest != idx:

            # Swap positions
            self.pos[ self.array[smallest][0] ] = idx
            self.pos[ self.array[idx][0] ] = smallest

            # Swap nodes
            self.swapMinHeapNode(smallest, idx)

            self.minHeapify(smallest)

    # Standard function to extract minimum
    # node from heap
    def extractMin(self):

        # Return NULL wif heap is empty
        if self.isEmpty() == True:
            return

        # Store the root node
        root = self.array[0]

        # Replace root node with last node
        lastNode = self.array[self.size - 1]
        self.array[0] = lastNode

        # Update position of last node
        self.pos[lastNode[0]] = 0
        self.pos[root[0]] = self.size - 1

        # Reduce heap size and heapify root
        self.size -= 1
        self.minHeapify(0)

        return root

    def isEmpty(self):
        return True if self.size == 0 else False

    def decreaseKey(self, v, dist):
```

```
        # Get the index of v in  heap array

        i = self.pos[v]

        # Get the node and update its dist value
        self.array[i][1] = dist

        # Travel up while the complete tree is
        # not hepified. This is a O(Logn) loop
        while i > 0 and self.array[i][1] < self.array[(i - 1) / 2][1]:

            # Swap this node with its parent
            self.pos[ self.array[i][0] ] = (i-1)/2
            self.pos[ self.array[(i-1)/2][0] ] = i
            self.swapMinHeapNode(i, (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
    def isInMinHeap(self, v):

        if self.pos[v] < self.size:
            return True
        return False


def printArr(dist, n):
    print "Vertex\tDistance from source"
    for i in range(n):
        print "%d\t\t%d" % (i,dist[i])


class Graph():

    def __init__(self, V):
        self.V = V
        self.graph = defaultdict(list)

    # Adds an edge to an undirected graph
    def addEdge(self, src, dest, 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. The first
        # element of the node has the destination
```

```
        # and the second elements has the weight
        newNode = [dest, weight]
        self.graph[src].insert(0, newNode)

        # Since graph is undirected, add an edge
        # from dest to src also
        newNode = [src, weight]
        self.graph[dest].insert(0, newNode)

    # The main function that calulates distances
    # of shortest paths from src to all vertices.
    # It is a O(ELogV) function
    def dijkstra(self, src):

        V = self.V  # Get the number of vertices in graph
        dist = []   # dist values used to pick minimum
                    # weight edge in cut

        # minHeap represents set E
        minHeap = Heap()

        #  Initialize min heap with all vertices.
        # dist value of all vertices
        for v in range(V):
            dist.append(sys.maxint)
            minHeap.array.append( minHeap.newMinHeapNode(v, dist[v]) )
            minHeap.pos.append(v)

        # Make dist value of src vertex as 0 so
        # that it is extracted first
        minHeap.pos[src] = src
        dist[src] = 0
        minHeap.decreaseKey(src, dist[src])

        # Initially size of min heap is equal to V
        minHeap.size = V;

        # In the following loop, min heap contains all nodes
        # whose shortest distance is not yet finalized.
        while minHeap.isEmpty() == False:

            # Extract the vertex with minimum distance value
            newHeapNode = minHeap.extractMin()
            u = newHeapNode[0]

            # Traverse through all adjacent vertices of
            # u (the extracted vertex) and update their
            # distance values
```

```
            for pCrawl in self.graph[u]:

                v = pCrawl[0]

                # If shortest distance to v is not finalized
                # yet, and distance to v through u is less
                # than its previously calculated distance
                if minHeap.isInMinHeap(v) and dist[u] != sys.maxint and \
                   pCrawl[1] + dist[u] < dist[v]:
                        dist[v] = pCrawl[1] + dist[u]

                        # update distance value
                        # in min heap also
                        minHeap.decreaseKey(v, dist[v])

        printArr(dist,V)


# Driver program to test the above functions
graph = Graph(9)
graph.addEdge(0, 1, 4)
graph.addEdge(0, 7, 8)
graph.addEdge(1, 2, 8)
graph.addEdge(1, 7, 11)
graph.addEdge(2, 3, 7)
graph.addEdge(2, 8, 2)
graph.addEdge(2, 5, 4)
graph.addEdge(3, 4, 9)
graph.addEdge(3, 5, 14)
graph.addEdge(4, 5, 10)
graph.addEdge(5, 6, 2)
graph.addEdge(6, 7, 1)
graph.addEdge(6, 8, 6)
graph.addEdge(7, 8, 7)
graph.dijkstra(0)

# This code is contributed by Divyanshu Mehta
```

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(LogV) time. So overall time complexity is O(E+V)*O(LogV) which is O((E+V)*LogV) = O(ELogV)
Note that the above code uses Binary Heap for Priority Queue implementation. Time complexity can be reduced to O(E + VLogV) using Fibonacci Heap. The reason is, Fibonacci Heap takes O(1) time for decrease-key operation while Binary Heap takes O(Logn) 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 dijekstra 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

Source

https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/

Chapter 17

Dijkstra's shortest path algorithm Greedy Algo-7

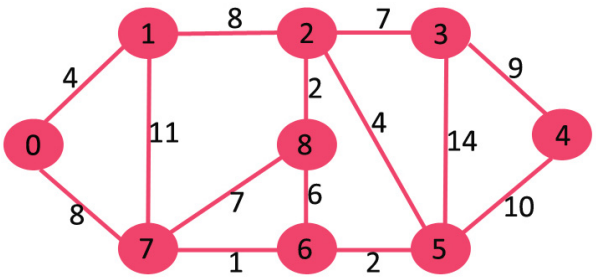
Dijsktra's algorithm

Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.

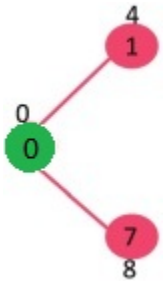
Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a *SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest 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 a minimum distance from the 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.

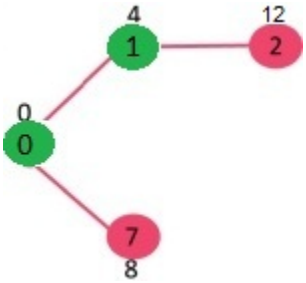
Let us understand with the following example:



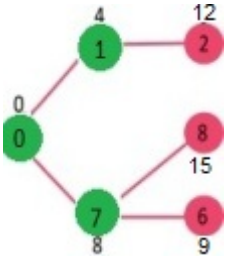
The set *sptSet* is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update 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 colour.



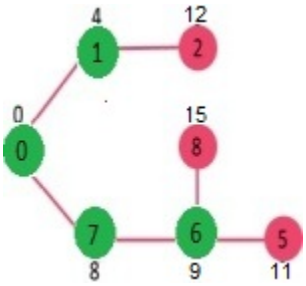
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



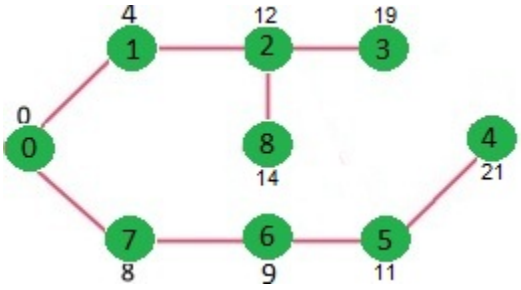
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



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 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.

C++

```
 // A 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 tt %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 the 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, 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;
}
```

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+" tt "+dist[i]);
}

// Funtion 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 stpSet[] 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
```

Python

```
 # Python program for Dijkstra's single
# source shortest path algorithm. The program is
# for adjacency matrix representation of the graph

# Library for INT_MAX
import sys

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print "Vertex tDistance from Source"
        for node in range(self.V):
            print node,"t",dist[node]

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minDistance(self, dist, sptSet):

        # Initilaize minimum distance for next node
        min = sys.maxint

        # Search not nearest vertex not in the
        # shortest path tree
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    # Funtion that implements Dijkstra's single source
    # shortest path algorithm for a graph represented
    # using adjacency matrix representation
    def dijkstra(self, src):

        dist = [sys.maxint] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
```

```
            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the
            # shotest path tree
            sptSet[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shotest path tree
            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and
                   dist[v] > dist[u] + self.graph[u][v]:
                        dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

# Driver program
g  = Graph(9)
g.graph = [[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]
          ];

g.dijkstra(0);

# This code is contributed by Divyanshu Mehta
```

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 the source to a single target, we can break the for the loop when the picked minimum distance vertex is equal to target (Step 3.a of the 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

Source

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Chapter 18

Divide 1 to n into two groups with minimum sum difference

Divide 1 to n into two groups with minimum sum difference - GeeksforGeeks

Given a positive integer n such that n > 2. Divide numbers from 1 to n in two groups such that absolute difference of sum of each group is minimum. Print any two groups with their size in first line and in next line print elements of that group.

Examples:

```
Input : 5
Output : 2
         5 2
         3
         4 3 1
Here sum of group 1 is 7 and sum of group 2 is 8.
Their absolute difference is 1 which is minimum.
We can have multiple correct answers. (1, 2, 5) and
(3, 4) is another such group.

Input : 6
Output : 2
         6 4
         4
         5 3 2 1
```

We can always divide sum of n integers in two groups such that their absolute difference of their sum is 0 or 1. So sum of group at most differ by 1. We define sum of group1 as half of n elements sum.

Now run a loop from n to 1 and insert i into group1 if inserting an element doesn't exceed

group1 sum otherwise insert that i into group2.
C++

```
 // CPP program to divide n integers
// in two groups such that absolute
// difference of their sum is minimum
#include <bits/stdc++.h>
using namespace std;

// To print vector along size
void printVector(vector<int> v)
{
    // Print vector size
    cout << v.size() << endl;

    // Print vector elements
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";

    cout << endl;
}

// To divide n in two groups such that
// absolute difference of their sum is
// minimum
void findTwoGroup(int n)
{
    // Find sum of all elements upto n
    int sum = n * (n + 1) / 2;

    // Sum of elements of group1
    int group1Sum = sum / 2;

    vector<int> group1, group2;

    for (int i = n; i > 0; i--) {

        // If sum is greater then or equal
        // to 0 include i in group 1
        // otherwise include in group2
        if (group1Sum - i >= 0) {

            group1.push_back(i);

            // Decrease sum of group1
            group1Sum -= i;
        }
        else {
```

```
            group2.push_back(i);
        }
    }

    // Print both the groups
    printVector(group1);
    printVector(group2);
}

// Driver program to test above functions
int main()
{
    int n = 5;
    findTwoGroup(n);
    return 0;
}
```

Java

```
 // Java program to divide n integers
// in two groups such that absolute
// difference of their sum is minimum
import java.io.*;
import java.util.*;

class GFG
{
    // To print vector along size
    static void printVector(Vector<Integer> v)
    {
        // Print vector size
        System.out.println(v.size());

        // Print vector elements
        for (int i = 0; i < v.size(); i++)
            System.out.print(v.get(i) + " ");

        System.out.println();
    }

    // To divide n in two groups such that
    // absolute difference of their sum is
    // minimum
    static void findTwoGroup(int n)
    {
        // Find sum of all elements upto n
        int sum = n * (n + 1) / 2;
```

```
        // Sum of elements of group1
        int group1Sum = sum / 2;

        Vector<Integer> group1 = new Vector<Integer>();
        Vector<Integer> group2 = new Vector<Integer>();

        for (int i = n; i > 0; i--) {

            // If sum is greater then or equal
            // to 0 include i in group1
            // otherwise include in group2
            if (group1Sum - i >= 0) {

                group1.add(i);

                // Decrease sum of group1
                group1Sum -= i;
            }
            else {
                group2.add(i);
            }
        }

        // Print both the groups
        printVector(group1);
        printVector(group2);
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 5;
        findTwoGroup(n);
    }
}

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

Python3

```
 # Python program to divide n integers
# in two groups such that absolute
# difference of their sum is minimum
import math

# To print vector along size
def printVector( v):
```

```
    # Print vector size
    print(len(v))

    # Print vector elements
    for i in range( 0, len(v)):
        print(v[i] , end =  " ")

    print()


# To divide n in two groups such that
# absolute difference of their sum is
# minimum
def findTwoGroup(n):

    # Find sum of all elements upto n
    sum = n * (n + 1) / 2

    # Sum of elements of group1
    group1Sum = sum / 2

    group1=[]
    group2=[]
    for i in range(n, 0, -1):

        # If sum is greater then or equal
        # to 0 include i in group 1
        # otherwise include in group2
        if (group1Sum - i >= 0) :
            group1.append(i)

            # Decrease sum of group1
            group1Sum -= i

        else :
            group2.append(i)

    # Print both the groups
    printVector(group1)
    printVector(group2)

# driver code
n = 5
findTwoGroup(n)

# This code is contributed by Gitanjali.
```

C#

```
// C# program to divide n integers
// in two groups such that absolute
// difference of their sum is minimum
using System;
using System.Collections;

class GFG
{
// To print vector along size
static void printVector(ArrayList v)
{
// Print vector size
Console.WriteLine(v.Count);

// Print vector elements
for (int i = 0; i < v.Count; i++) Console.Write(v[i] + " "); Console.WriteLine(); } // To
divide n in two groups // such that absolute difference // of their sum is minimum static
void findTwoGroup(int n) { // Find sum of all elements upto n int sum = n * (n + 1) / 2; //
Sum of elements of group1 int group1Sum = sum / 2; ArrayList group1 = new ArrayList();
ArrayList group2 = new ArrayList(); for (int i = n; i > 0; i–)
{

// If sum is greater then
// or equal to 0 include i
// in group1 otherwise
// include in group2
if (group1Sum – i >= 0)
{
group1.Add(i);

// Decrease sum of group1
group1Sum -= i;
}
else
{
group2.Add(i);
}
}

// Print both the groups
printVector(group1);
printVector(group2);
}

// Driver code
public static void Main()
{
int n = 5;
findTwoGroup(n);
}
}
```

// This code is contributed by mits

PHP

```
 <?php
// PHP program to divide n
// integers in two groups
// such that absolute
// difference of their
// sum is minimum

// To print vector
// along size
function printVector($v)
{
    // Print vector size
    echo count($v) . "\n";

    // Print vector elements
    for ($i = 0;
         $i < count($v); $i++)
        echo $v[$i] . " ";

    echo "\n";
}

// To divide n in two groups
// such that absolute difference
// of their sum is minimum
function findTwoGroup($n)
{
    // Find sum of all
    // elements upto n
    $sum = $n * ($n + 1) / 2;

    // Sum of elements
    // of group1
    $group1Sum = (int)($sum / 2);

    $group1;
    $group2;
    $x = 0;
    $y = 0;

    for ($i = $n; $i > 0; $i--)
    {

        // If sum is greater then
        // or equal to 0 include
```

```
        // i in group 1 otherwise
        // include in group2
        if ($group1Sum - $i >= 0)
        {

            $group1[$x++] = $i;

            // Decrease sum
            // of group1
            $group1Sum -= $i;
        }
        else
        {
            $group2[$y++] = $i;
        }
    }

    // Print both the groups
    printVector($group1);
    printVector($group2);
}

// Driver Code
$n = 5;
findTwoGroup($n);

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

Output:

```
2
5 2
3
4 3 1
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/divide-1-n-two-groups-minimum-sum-difference/

Chapter 19

Divide cuboid into cubes such that sum of volumes is maximum

Divide cuboid into cubes such that sum of volumes is maximum - GeeksforGeeks

Given the **length**, **breadth**, **height** of a cuboid. The task is to divide the given cuboid in minimum number of cubes such that size of all cubes is same and sum of volumes of cubes is maximum.

Examples:

```
Input : l = 2, b = 4, h = 6
Output : 2 6
A cuboid of length 2, breadth 4 and
height 6 can be divided into 6 cube
of side equal to 2.
Volume of cubes = 6*(2*2*2) = 6*8 = 48.
Volume of cuboid = 2*4*6 = 48.

Input : 1 2 3
Output : 1 6
```

First of all, we are not allowed to waste volume of cuboid as we meed maximum volume sum. So, each side should be completely divide among all cubes. And since each of three side of cubes are equal, so each side of the cuboid need to be divisible by same number, say x, which will going to be the side of the cube. So, we have to maximize this x, which will divide given length, breadth and height. This x will be maximum only if it is greatest common divisor of given length, breadth and height. So, the length of the cube will be GCD of length, breadth and height.

Now, to compute number of cubes, we know total volume of cuboid and can find volume of one cube (since side is already calculated). So, total number of cubes is equal to (volume of cuboid)/(volume of cube) i.e (l * b * h)/(x * x * x).

Below is implementation of this approach:
C++

```
 // CPP program to find optimal way to break
// cuboid into cubes.
#include <bits/stdc++.h>
using namespace std;

// Print the maximum side and no of cube.
void maximizecube(int l, int b, int h)
{
    // GCD to find side.
    int side = __gcd(l, __gcd(b, h));

    // dividing to find number of cubes.
    int num = l / side;
    num = (num * b / side);
    num = (num * h / side);

    cout << side << " " << num << endl;
}

// Driver code
int main()
{
    int l = 2, b = 4, h = 6;

    maximizecube(l, b, h);
    return 0;
}
```

Java

```
 // JAVA Code for Divide cuboid into cubes
// such that sum of volumes is maximum
import java.util.*;

class GFG {

    static int gcd(int m, int n)
    {
        if(n == 0)
            return m;
        else if(n > m)
```

```
            return gcd(n,m);
        else
            return gcd(n, m % n);
    }

    // Print the maximum side and no
    //     of cube.
    static void maximizecube(int l, int b,
                                    int h)
    {
        // GCD to find side.
        int side = gcd(l, gcd(b, h));

        // dividing to find number of cubes.
        int num = l / side;
        num = (num * b / side);
        num = (num * h / side);

       System.out.println( side + " " + num);
    }

    /* Driver program  */
    public static void main(String[] args)
    {
         int l = 2, b = 4, h = 6;
         maximizecube(l, b, h);
    }
}

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

Python3

```
 # Python3 code to find optimal way to break
# cuboid into cubes.
from fractions import gcd

# Print the maximum side and no of cube.
def maximizecube( l , b , h ):

    # GCD to find side.
    side = gcd(l, gcd(b, h))

    # dividing to find number of cubes.
    num = int(l / side)
    num = int(num * b / side)
    num = int(num * h / side)
```

```
    print(side, num)

# Driver code
l = 2
b = 4
h = 6

maximizecube(l, b, h)

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

C#

```
 // C# Code for Divide cuboid into cubes
// such that sum of volumes is maximum
using System;

class GFG {

    static int gcd(int m, int n)
    {
        if(n == 0)
            return m;
        else if(n > m)
            return gcd(n,m);
        else
            return gcd(n, m % n);
    }

    // Print the maximum side and no
    // of cube.
    static void maximizecube(int l, int b,
                                    int h)
    {
        // GCD to find side.
        int side = gcd(l, gcd(b, h));

        // dividing to find number of cubes.
        int num = l / side;
        num = (num * b / side);
        num = (num * h / side);

    Console.WriteLine( side + " " + num);
    }

    /* Driver program */
    public static void Main()
    {
```

```
        int l = 2, b = 4, h = 6;
        maximizecube(l, b, h);
    }
}

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

PHP

```
 <?php
// PHP program to find optimal way
// to break cuboid into cubes.

// Recursive function to
// return gcd of a and b
function __gcd($a, $b)
{

    // Everything divides 0
    if($a == 0 or $b == 0)
        return 0 ;

    // base case
    if($a == $b)
        return $a ;

    // a is greater
    if($a > $b)
        return __gcd($a - $b , $b ) ;

    return __gcd($a , $b - $a) ;
}


// Print the maximum side and no of cube.
function maximizecube($l, $b, $h)
{

    // GCD to find side.
    $side = __gcd($l, __gcd($b, $h));

    // dividing to find number of cubes.
    $num = $l / $side;
    $num = ($num * $b / $side);
    $num = ($num * $h / $side);

    echo $side , " " , $num ;
}
```

```
    // Driver code
    $l = 2;
    $b = 4;
    $h = 6;
    maximizecube($l, $b, $h);

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

Output:

```
2 6
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/divide-cuboid-cubes-sum-volumes-maximum/

Chapter 20

Efficient Huffman Coding for Sorted Input Greedy Algo-4

Efficient Huffman Coding for Sorted Input Greedy Algo-4 - GeeksforGeeks

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(nLogn). 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 from 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;
}
```

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.

Source

https://www.geeksforgeeks.org/efficient-huffman-coding-for-sorted-input-greedy-algo-4/

Chapter 21

Final cell position in the matrix

Final cell position in the matrix - GeeksforGeeks

Given an array of list of commands U(Up), D(Down), L(Left) and R(Right) and initial cell position (x, y) in a matrix. Find the final cell position of the object in the matrix after following the given commands. It is assumed that the final required cell position exists in the matrix.

Examples:

```
Input : command[] = "DDLRULL"
        x = 3, y = 4
Output : (1, 5)

Input : command[] = "LLRUUUDRRDDDULRLLUDUUR"
        x = 6, y = 5
Output : (6, 3)
```

Source: Flipkart Interview (SDE-1 On Campus).

Approach: Following are the steps:

1. Count cup, cdown, cleft and cright for U(Up), D(Down), L(Left) and R(Right) movements respectively.
2. Calculate final_x = x + (cright – cleft) and final_y = y + (cdown – cup).

The final cell position is (final_x, final_y)

C++

```
 // C++ implementation to find the final cell position
// in the given matrix
```

```
#include <bits/stdc++.h>

using namespace std;

// function to find the final cell position
// in the given matrix
void finalPos(char command[], int n,
              int x, int y)
{
    // to count up, down, left and cright
    // movements
    int cup, cdown, cleft, cright;

    // to store the final coordinate position
    int final_x, final_y;

    cup = cdown = cleft = cright = 0;

    // traverse the command array
    for (int i = 0; i < n; i++) {
        if (command[i] == 'U')
            cup++;
        else if (command[i] == 'D')
            cdown++;
        else if (command[i] == 'L')
            cleft++;
        else if (command[i] == 'R')
            cright++;
    }

    // calculate final values
    final_x = x + (cright - cleft);
    final_y = y + (cdown - cup);

    cout << "Final Position: "
         << "("
         << final_x << ", " << final_y << ")";
}

// Driver program to test above
int main()
{
    char command[] = "DDLRULL";
    int n = (sizeof(command) / sizeof(char)) - 1;
    int x = 3, y = 4;
    finalPos(command, n, x, y);
    return 0;
}
```

Java

```
 // Java implementation to find
// the final cell position
// in the given matrix
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
// function to find the
// final cell position
// in the given matrix
static void finalPos(String command, int n,
                             int x, int y)
{
    // to count up, down, left
    // and cright movements
    int cup, cdown, cleft, cright;

    // to store the final
    // coordinate position
    int final_x, final_y;

    cup = cdown = cleft = cright = 0;

    // traverse the command array
    for (int i = 0; i < n; i++)
    {
        if (command.charAt(i) == 'U')
            cup++;
        else if (command.charAt(i) == 'D')
            cdown++;
        else if (command.charAt(i)== 'L')
            cleft++;
        else if (command.charAt(i) == 'R')
            cright++;
    }

    // calculate final values
    final_x = x + (cright - cleft);
    final_y = y + (cdown - cup);

    System.out.println("Final Position: " +
                       "(" + final_x + ", " +
                              final_y + ")");
}
```

```
// Driver Code
public static void main(String []args)
{
    String command = "DDLRULL";
    int n = command.length();
    int x = 3, y = 4;
    finalPos(command, n, x, y);
}
}

// This code is contributed
// by Subhadeep
```

Output:

```
Final Position: (1, 5)
```

Time Complexity: O(n), where n is the number of commands.

Improved By : tufan_gupta2000

Source

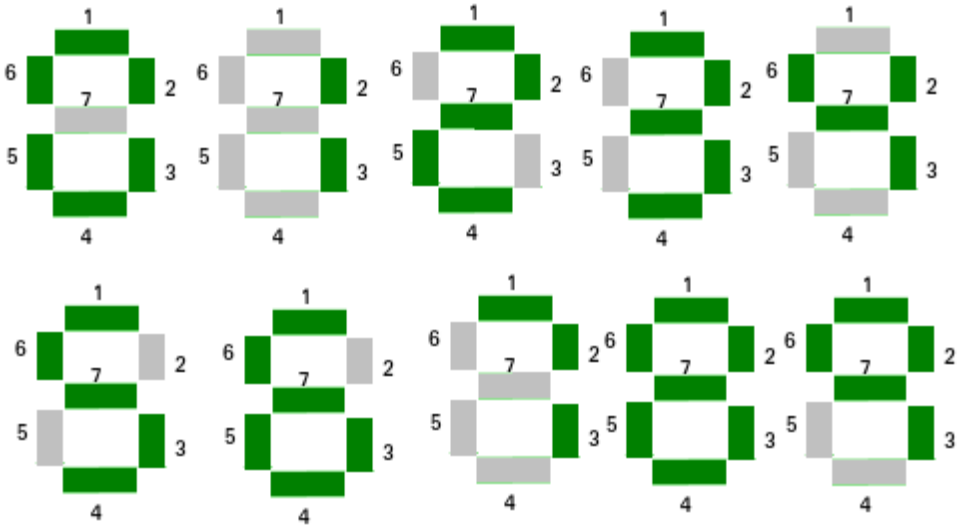
https://www.geeksforgeeks.org/final-cell-position-in-the-matrix/

Chapter 22

Find element using minimum segments in Seven Segment Display

Find element using minimum segments in Seven Segment Display - GeeksforGeeks

A Seven Segment Display can be used to to display numbers. Given an array of **n** natural numbers. The task is to find the number in the array which is using minimum segments to display number. If multiple numbers have a minimum number of segments, output the number having the smallest index.



Examples :

```
Input : arr[] = { 1, 2, 3, 4, 5 }.
Output : 1
1 uses on 2 segments to display.

Input : arr[] = { 489, 206, 745, 123, 756 }.
Output : 745
```

Precompute the number of segment used by digits from 0 to 9 and store it. Now for each element of the array sum the number of segment used by each digit. Then find the element which is using the minimum number of segments.

The number of segment used by digit:
0 -> 6
1 -> 2
2 -> 5
3 -> 5
4 -> 4
5 -> 5
6 -> 6
7 -> 3
8 -> 7
9 -> 6

Below is the implementation of this approach:
C++

```
 // C++ program to find minimum number of segments
// required
#include<bits/stdc++.h>
using namespace std;

// Precomputed values of segment used by digit 0 to 9.
const int seg[10] = { 6, 2, 5, 5, 4, 5, 6, 3, 7, 6};

// Return the number of segments used by x.
int computeSegment(int x)
{
    if (x == 0)
        return seg[0];

    int count = 0;

    // Finding sum of the segment used by
    // each digit of a number.
    while (x)
    {
        count += seg[x%10];
        x /= 10;
```

```
    }

    return count;
}

int elementMinSegment(int arr[], int n)
{
    // Initalising the minimum segment and minimum
    // number index.
    int minseg = computeSegment(arr[0]);
    int minindex = 0;

    // Finding and comparing segment used
    // by each number arr[i].
    for (int i = 1; i < n; i++)
    {
        int temp = computeSegment(arr[i]);

        // If arr[i] used less segment then update
        // minimum segment and minimum number.
        if (temp < minseg)
        {
            minseg   = temp;
            minindex = i;
        }
    }

    return arr[minindex];
}

// Driven Program
int main()
{
    int arr[] = {489, 206, 745, 123, 756};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << elementMinSegment(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to find minimum
// number of segments required
import java.io.*;

class GFG {

// Precomputed values of segment
```

```
// used by digit 0 to 9.
static int []seg = { 6, 2, 5, 5, 4, 5, 6, 3, 7, 6};

// Return the number of segments used by x.
static int computeSegment(int x)
{
    if (x == 0)
        return seg[0];

    int count = 0;

    // Finding sum of the segment used by
    // each digit of a number.
    while (x > 0)
    {
        count += seg[x % 10];
        x /= 10;
    }

    return count;
}

static int elementMinSegment(int []arr, int n)
{
    // Initalising the minimum segment
    // and minimum number index.
    int minseg = computeSegment(arr[0]);
    int minindex = 0;

    // Finding and comparing segment used
    // by each number arr[i].
    for (int i = 1; i < n; i++)
    {
        int temp = computeSegment(arr[i]);

        // If arr[i] used less segment then update
        // minimum segment and minimum number.
        if (temp < minseg)
        {
            minseg = temp;
            minindex = i;
        }
    }

    return arr[minindex];
}

    // Driver program
```

```
    static public void main (String[] args)
    {
       int []arr = {489, 206, 745, 123, 756};
       int n = arr.length;
       System.out.println(elementMinSegment(arr, n));
    }
}

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

C#

```
 // C# program to find minimum
// number of segments required
using System;

class GFG{

// Precomputed values of segment
// used by digit 0 to 9.
static int []seg = new int[10]{ 6, 2, 5, 5, 4,
                                5, 6, 3, 7, 6};

// Return the number of segments used by x.
static int computeSegment(int x)
{
    if (x == 0)
        return seg[0];

    int count = 0;

    // Finding sum of the segment used by
    // each digit of a number.
    while (x > 0)
    {
        count += seg[x % 10];
        x /= 10;
    }

    return count;
}

static int elementMinSegment(int []arr, int n)
{
    // Initalising the minimum segment
    // and minimum number index.
    int minseg = computeSegment(arr[0]);
    int minindex = 0;
```

```
    // Finding and comparing segment used
    // by each number arr[i].
    for (int i = 1; i < n; i++)
    {
        int temp = computeSegment(arr[i]);

        // If arr[i] used less segment then update
        // minimum segment and minimum number.
        if (temp < minseg)
        {
            minseg = temp;
            minindex = i;
        }
    }

    return arr[minindex];
}

    // Driver program
    static public void Main()
    {
       int []arr = {489, 206, 745, 123, 756};
       int n = arr.Length;
       Console.WriteLine(elementMinSegment(arr, n));
    }
}

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

PHP

```
 <?php
// PHP program to find minimum
// number of segments required

// Precomputed values of segment
// used by digit 0 to 9.

$seg = array(6, 2, 5, 5, 4,
             5, 6, 3, 7, 6);

// Return the number of
// segments used by x.
function computeSegment($x)
{
    global $seg;
    if ($x == 0)
```

```
        return $seg[0];

    $count = 0;

    // Finding sum of the segment
    // used by each digit of a number.
    while ($x)
    {
        $count += $seg[$x % 10];
        $x = (int)$x / 10;
    }

    return $count;
}

function elementMinSegment($arr, $n)
{
    // Initalising the minimum segment
    // and minimum number index.
    $minseg = computeSegment($arr[0]);
    $minindex = 0;

    // Finding and comparing segment
    // used by each number arr[i].
    for ($i = 1; $i < $n; $i++)
    {
        $temp = computeSegment($arr[$i]);

        // If arr[i] used less segment
        // then update minimum segment
        // and minimum number.
        if ($temp < $minseg)
        {
            $minseg = $temp;
            $minindex = $i;
        }
    }

    return $arr[$minindex];
}

// Driver Code
$arr = array (489, 206, 745, 123, 756);
$n = sizeof($arr);
echo elementMinSegment($arr, $n) ,"\n";

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

Output :

```
745
```

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/find-element-using-minimum-segments-seven-segment-display/

Chapter 23

Find if k bookings possible with given arrival and departure times

Find if k bookings possible with given arrival and departure times - GeeksforGeeks

A hotel manager has to process N advance bookings of rooms for the next season. His hotel has K rooms. Bookings contain an arrival date and a departure date. He wants to find out whether there are enough rooms in the hotel to satisfy the demand.

The idea is to sort the arrays and keep track of overlaps.

Examples:

```
Input :  Arrivals :   [1 3 5]
         Departures : [2 6 8]
         K : 1
Output : False
Hotel manager needs at least two
rooms as the second and third
intervals overlap.
```

The idea is store arrival and departure times in an auxiliary array with an additional marker to indicate whether time is arrival or departure. Now sort the array. Process the sorted array, for every arrival increment active bookings. And for every departure, decrement. Keep track of maximum active bookings. If count of active bookings at any moment is more than k, then return false. Else return return true.

Below is the c++ code for above approach.

```
#include <bits/stdc++.h>
```

```
using namespace std;

bool areBookingsPossible(int arrival[],
           int departure[], int n, int k)
{
    vector<pair<int, int> > ans;

    // create a common vector both arrivals
    // and departures.
    for (int i = 0; i < n; i++) {
        ans.push_back(make_pair(arrival[i], 1));
        ans.push_back(make_pair(departure[i], 0));
    }

    // sort the vector
    sort(ans.begin(), ans.end());

    int curr_active = 0, max_active = 0;

    for (int i = 0; i < ans.size(); i++) {

        // if new arrival, increment current
        // guests count and update max active
        // guests so far
        if (ans[i].second == 1) {
            curr_active++;
            max_active = max(max_active,
                             curr_active);
        }

        // if a guest departs, decrement
        // current guests count.
        else
            curr_active--;
    }

    // if max active guests at any instant
    // were more than the available rooms,
    // return false. Else return true.
    return (k >= max_active);
}

int main()
{
    int arrival[] = { 1, 3, 5 };
    int departure[] = { 2, 6, 8 };
    int n = sizeof(arrival) / sizeof(arrival[0]);
    cout << (areBookingsPossible(arrival,
```

```
            departure, n, 1) ? "Yes\n" : "No\n");
    return 0;
}
```

Output:

```
No
```

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

Source

https://www.geeksforgeeks.org/find-k-bookings-possible-given-arrival-departure-times/

Chapter 24

Find k pairs with smallest sums in two arrays Set 2

Find k pairs with smallest sums in two arrays Set 2 - GeeksforGeeks

Given two arrays arr1[] and arr2[] sorted in ascending order and an integer K. The task is to find k pairs with smallest sums such that one element of a pair belongs to arr1[] and another element belongs to arr2[]. Sizes of arrays may be different. Assume all the elements to be distinct in each array.

Examples:

```
Input: a1[] = {1, 7, 11}
       a2[] = {2, 4, 6}
       k = 3
Output: [1, 2], [1, 4], [1, 6]
The first 3 pairs are returned
from the sequence [1, 2], [1, 4], [1, 6], [7, 2],
[7, 4], [11, 2], [7, 6], [11, 4], [11, 6].

Input: a1[] = { 2, 3, 4 }
       a2[] = { 1, 6, 5, 8 }
       k = 4
Output: [1, 2] [1, 3] [1, 4] [2, 6]
```

An approach with time complexity O(k*n1) has been discussed here.

Efficient Approach: Since the array is already sorted. The given below algorithm can be followed to solve this problem:

- The idea is to maintain two pointers, one pointer pointing to one pair in (a1, a2) and other in (a2, a1). Each time, compare the sums of the elements pointed by the two

pairs and print the minimum one. After this, increment the pointer to the element in the printed pair which was larger than the other. This helps to get the next possible k smallest pair.

- Once the pointer has been updated to the element such that it starts pointing to the first element of the array again, update the other pointer to the next value. This update is done cyclically.
- Also, when both the pairs are pointing to the same element, update pointers in both the pairs so as to avoid extra pair's printing. Update one pair's pointer according to rule1 and other's opposite to rule1. This is done to ensure that all the permutations are considered and no repetitions of pairs are there.

Below is the working of the algorithm for example 1:

a1[] = {1, 7, 11}, a2[] = {2, 4}, k = 3

Let the pairs of pointers be _one, _two

_one.first points to 1, _one.second points to 2 ;
_two.first points to 2, _two.second points to 1

1st pair:
Since _one and _two are pointing to same elements, print the pair once and update

- print [1, 2]

then update _one.first to 1, _one.second to 4 (following rule 1) ;
_two.first points to 2, _two.second points to 7 (opposite to rule 1).
If rule 1 was followed for both, then both of them would have been pointing to 1 and 4,
and it is not possible to get all possible permutations.

2nd pair:
Since a1[_one.first] + a2[_one.second] < a1[_two.second] + a2[_two.first], print them and update

- print [1, 4]

then update _one.first to 1, _one.second to 2
Since _one.second came to the first element of the array once again,
therefore _one.first points to 7

Repeat the above process for remaining K pairs

Below is the C++ implementation of the above approach:

```
 // C++ program to print the k smallest
// pairs | Set 2
#include <bits/stdc++.h>
using namespace std;
```

```
typedef struct _pair {
    int first, second;
} _pair;

// Functio to print the K smallest pairs
void printKPairs(int a1[], int a2[],
                 int size1, int size2, int k)
{

    // if k is greater than total pairs
    if (k > (size2 * size1)) {
        cout << "k pairs don't exist\n";
        return;
    }

    // _pair _one keeps track of
    // 'first' in a1 and 'second' in a2
    // in _two, _two.first keeps track of
    // element in the a2[] and _two.second in a1[]
    _pair _one, _two;
    _one.first = _one.second = _two.first = _two.second = 0;

    int cnt = 0;

    // Repeat the above process till
    // all K pairs are printed
    while (cnt < k) {

        // when both the pointers are pointing
        // to the same elements (point 3)
        if (_one.first == _two.second
            && _two.first == _one.second) {
            if (a1[_one.first] < a2[_one.second]) {
                cout << "[" << a1[_one.first]
                     << ", " << a2[_one.second] << "] ";

                // updates according to step 1
                _one.second = (_one.second + 1) % size2;
                if (_one.second == 0) // see point 2
                    _one.first = (_one.first + 1) % size1;

                // updates opposite to step 1
                _two.second = (_two.second + 1) % size2;
                if (_two.second == 0)
                    _two.first = (_two.first + 1) % size2;
            }
            else {
```

```
            cout << "[" << a2[_one.second]
                 << ", " << a1[_one.first] << "] ";

            // updates according to rule 1
            _one.first = (_one.first + 1) % size1;
            if (_one.first == 0) // see point 2
                _one.second = (_one.second + 1) % size2;

            // updates opposite to rule 1
            _two.first = (_two.first + 1) % size2;
            if (_two.first == 0) // see point 2
                _two.second = (_two.second + 1) % size1;
        }
    }
    // else update as necessary (point 1)
    else if (a1[_one.first] + a2[_one.second]
             <= a2[_two.first] + a1[_two.second]) {
        if (a1[_one.first] < a2[_one.second]) {
            cout << "[" << a1[_one.first] << ", "
                 << a2[_one.second] << "] ";

            // updating according to rule 1
            _one.second = ((_one.second + 1) % size2);
            if (_one.second == 0) // see point 2
                _one.first = (_one.first + 1) % size1;
        }
        else {
            cout << "[" << a2[_one.second] << ", "
                 << a1[_one.first] << "] ";

            // updating according to rule 1
            _one.first = ((_one.first + 1) % size1);
            if (_one.first == 0) // see point 2
                _one.second = (_one.second + 1) % size2;
        }
    }
    else if (a1[_one.first] + a2[_one.second]
             > a2[_two.first] + a1[_two.second]) {
        if (a2[_two.first] < a1[_two.second]) {
            cout << "[" << a2[_two.first] << ", " << a1[_two.second] << "] ";

            // updating according to rule 1
            _two.first = ((_two.first + 1) % size2);
            if (_two.first == 0) // see point 2
                _two.second = (_two.second + 1) % size1;
        }
        else {
            cout << "[" << a1[_two.second]
```

```
                    << ", " << a2[_two.first] << "] ";

                // updating according to rule 1
                _two.second = ((_two.second + 1) % size1);
                if (_two.second == 0) // see point 2
                    _two.first = (_two.first + 1) % size1;
            }
        }
        cnt++;
    }
}

// Driver Code
int main()
{

    int a1[] = { 2, 3, 4 };
    int a2[] = { 1, 6, 5, 8 };
    int size1 = sizeof(a1) / sizeof(a1[0]);
    int size2 = sizeof(a2) / sizeof(a2[0]);
    int k = 4;
    printKPairs(a1, a2, size1, size2, k);
    return 0;
}
```

Output:

```
[1, 2] [1, 3] [1, 4] [2, 6]
```

Time complexity: O(K)

Source

https://www.geeksforgeeks.org/find-k-pairs-with-smallest-sums-in-two-arrays-set-2/

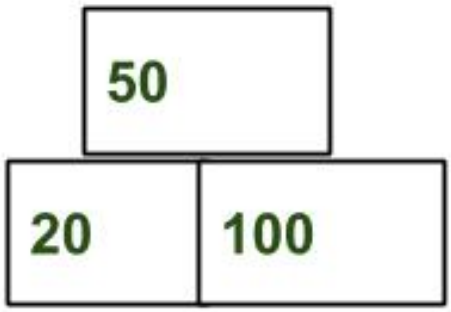
Chapter 25

Find maximum height pyramid from the given array of objects

Find maximum height pyramid from the given array of objects - GeeksforGeeks

Given **n** objects, with each object has width $\mathbf{w_i}$. We need to arrange them in a pyramidal way such that :

1. Total width of i^{th} is less than $(i + 1)^{th}$.
2. Total number of objects in the i^{th} is less than $(i + 1)^{th}$.



The task is to find the maximum height that can be achieved from given objects.

Examples :

```
Input : arr[] = {40, 100, 20, 30}
Output : 2
Top level : 30.
Lower (or bottom) level : 20, 40 and 100
Other possibility can be placing
20 on the top, and at second level any
other 4 objects. Another possibility is
to place 40 at top and other three at the
bottom.

Input : arr[] = {10, 20, 30, 50, 60, 70}
Output : 3
```

The idea is to use greedy approach by placing the object with the lowest width at the top, the next object at the level right below and so on.
To find the maximum number of levels, sort the given array and try to form pyramid from top to bottom. Find the smallest element of array i.e first element of array after sorting, place it on the top. Then try to build levels below it with greater number of objects and greater width.

Below is the implementation of this approach:
C++

```
 // C++ program to find maximum height pyramid
// from the given object width.
#include<bits/stdc++.h>
using namespace std;

// Returns maximum number of pyramidcal levels
// n boxes of given widths.
int maxLevel(int boxes[], int n)
{
    // Sort objects in increasing order of widths
    sort(boxes, boxes + n);

    int ans = 1;  // Initialize result

    // Total width of previous level and total
    // number of objects in previous level
    int prev_width = boxes[0];
    int prev_count = 1;

    // Number of object in current level.
```

```
    int curr_count = 0;

    // Width of current level.
    int curr_width = 0;
    for (int i=1; i<n; i++)
    {
        // Picking the object. So increase current
        // width and number of object.
        curr_width += boxes[i];
        curr_count += 1;

        // If current width and number of object
        // are greater than previous.
        if (curr_width > prev_width &&
            curr_count > prev_count)
        {
            // Update previous width, number of
            // object on previous level.
            prev_width = curr_width;
            prev_count = curr_count;

            // Reset width of current level, number
            // of object on current level.
            curr_count = 0;
            curr_width = 0;

            // Increment number of level.
            ans++;
        }
    }

    return ans;
}

// Driver Program
int main()
{
    int boxes[] = {10, 20, 30, 50, 60, 70};
    int n = sizeof(boxes)/sizeof(boxes[0]);
    cout << maxLevel(boxes, n) << endl;
    return 0;
}
```

Java

```
 // Java program to find maximum height pyramid
// from the given object width.
import java.io.*;
```

```
import java.util.Arrays;

class GFG {

    // Returns maximum number of pyramidcal
    // levels n boxes of given widths.
    static int maxLevel(int []boxes, int n)
    {

        // Sort objects in increasing order
        // of widths
        Arrays.sort(boxes);

        int ans = 1; // Initialize result

        // Total width of previous level
        // and total number of objects in
        // previous level
        int prev_width = boxes[0];
        int prev_count = 1;

        // Number of object in current
        // level.
        int curr_count = 0;

        // Width of current level.
        int curr_width = 0;
        for (int i = 1; i < n; i++)
        {
            // Picking the object. So
            // increase current width
            // and number of object.
            curr_width += boxes[i];
            curr_count += 1;

            // If current width and
            // number of object
            // are greater than previous.
            if (curr_width > prev_width &&
                curr_count > prev_count)
            {

                // Update previous width,
                // number of object on
                // previous level.
                prev_width = curr_width;
                prev_count = curr_count;
```

```
                // Reset width of current
                // level, number of object
                // on current level.
                curr_count = 0;
                curr_width = 0;

                // Increment number of
                // level.
                ans++;
            }
        }

        return ans;
    }

    // Driver Program
    static public void main (String[] args)
    {
        int []boxes = {10, 20, 30, 50, 60, 70};
        int n = boxes.length;
        System.out.println(maxLevel(boxes, n));
    }
}

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

Python 3

```
 # Python 3 program to find
# maximum height pyramid from
# the given object width.

# Returns maximum number
# of pyramidcal levels n
# boxes of given widths.
def maxLevel(boxes, n):

    # Sort objects in increasing
    # order of widths
    boxes.sort()

    ans = 1 # Initialize result

    # Total width of previous
    # level and total number of
    # objects in previous level
    prev_width = boxes[0]
    prev_count = 1
```

```
    # Number of object in
    # current level.
    curr_count = 0

    # Width of current level.
    curr_width = 0
    for i in range(1, n):

        # Picking the object. So
        # increase current width
        # and number of object.
        curr_width += boxes[i]
        curr_count += 1

        # If current width and
        # number of object are
        # greater than previous.
        if (curr_width > prev_width and
            curr_count > prev_count):

            # Update previous width,
            # number of object on
            # previous level.
            prev_width = curr_width
            prev_count = curr_count

            # Reset width of current
            # level, number of object
            # on current level.
            curr_count = 0
            curr_width = 0

            # Increment number of level.
            ans += 1
    return ans

# Driver Code
if __name__ == "__main__":
    boxes= [10, 20, 30, 50, 60, 70]
    n = len(boxes)
    print(maxLevel(boxes, n))

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# program to find maximum height pyramid
// from the given object width.
using System;

public class GFG {

    // Returns maximum number of pyramidcal
    // levels n boxes of given widths.
    static int maxLevel(int []boxes, int n)
    {
        // Sort objects in increasing order
        // of widths
        Array.Sort(boxes);

        int ans = 1; // Initialize result

        // Total width of previous level
        // and total number of objects in
        // previous level
        int prev_width = boxes[0];
        int prev_count = 1;

        // Number of object in current
        // level.
        int curr_count = 0;

        // Width of current level.
        int curr_width = 0;
        for (int i=1; i<n; i++)
        {
            // Picking the object. So
            // increase current width
            // and number of object.
            curr_width += boxes[i];
            curr_count += 1;

            // If current width and
            // number of object
            // are greater than previous.
            if (curr_width > prev_width &&
                   curr_count > prev_count)
            {

                // Update previous width,
                // number of object on
                // previous level.
                prev_width = curr_width;
                prev_count = curr_count;
```

```
                // Reset width of current
                // level, number of object
                // on current level.
                curr_count = 0;
                curr_width = 0;

                // Increment number of
                // level.
                ans++;
            }
        }

        return ans;
    }

    // Driver Program
    static public void Main ()
    {
        int []boxes = {10, 20, 30, 50, 60, 70};
        int n = boxes.Length;
        Console.WriteLine(maxLevel(boxes, n));
    }
}

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

PHP

```
 <?php
// PHP program to find maximum
// height pyramid from the
// given object width.

// Returns maximum number of
// pyramidcal levels n boxes
// of given widths.
function maxLevel($boxes, $n)
{
    // Sort objects in increasing
    // order of widths
    sort($boxes);

    // Initialize result
    $ans = 1;

    // Total width of previous
    // level and total number
```

```
    // of objects in previous level
    $prev_width = $boxes[0];
    $prev_count = 1;

    // Number of object
    // in current level.
    $curr_count = 0;

    // Width of current level.
    $curr_width = 0;
    for ( $i = 1; $i < $n; $i++)
    {
        // Picking the object. So
        // increase current width
        // and number of object.
        $curr_width += $boxes[$i];
        $curr_count += 1;

        // If current width and number
        // of object are greater
        // than previous.
        if ($curr_width > $prev_width and
            $curr_count > $prev_count)
        {
            // Update previous width, number
            // of object on previous level.
            $prev_width = $curr_width;
            $prev_count = $curr_count;

            // Reset width of current
            // level, number of object
            // on current level.
            $curr_count = 0;
            $curr_width = 0;

            // Increment number of level.
            $ans++;
        }
    }
    return $ans;
}

// Driver Code
$boxes = array(10, 20, 30, 50, 60, 70);
$n = count($boxes);
echo maxLevel($boxes, $n) ;

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

```
?>
```

Output :

```
3
```

Time Complexity : O(n log n).

Please see below article for more efficient solutions.
Maximum height of triangular arrangement of array values

Improved By : vt_m, ChitraNayal

Source

https://www.geeksforgeeks.org/find-maximum-height-pyramid-from-the-given-array-of-objects/

Chapter 26

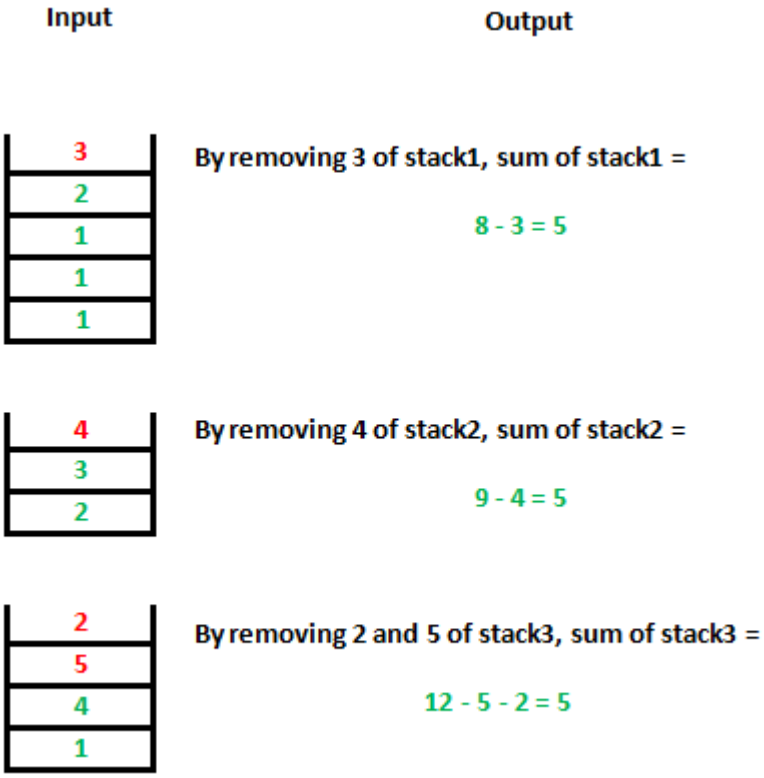
Find maximum sum possible equal sum of three stacks

Find maximum sum possible equal sum of three stacks - GeeksforGeeks

Given three stack of the positive numbers, the task is to find the possible equal maximum sum of the stacks with removal of top elements allowed. Stacks are represented as array, and the first index of the array represent the top element of the stack.

Examples:

```
Input : stack1[] = { 3, 10}
  stack2[] = { 4, 5 }
  stack3[] = { 2, 1 }
Output : 0
Sum can only be equal after removing all elements
from all stacks.
```



The idea is to compare the sum of each stack, if they are not same, remove the top element of the stack having the maximum sum.

Algorithm for solving this problem:

1. Find sum of all elements of in individual stacks.
2. If the sum of all three stacks is same, then this is the maximum sum.
3. Else remove the top element of the stack having the maximum sum among three of stacks. Repeat step 1 and step 2.

The approach works because elements are positive. To make sum equal, we must remove some element from stack having more sum and we can only remove from top.

Below is the implementation of this approach:
C/C++

```
 // C++ program to calculate maximum sum with equal
// stack sum.
#include<bits/stdc++.h>
using namespace std;
```

```
// Returns maximum possible equal sum of three stacks
// with removal of top elements allowed
int maxSum(int stack1[], int stack2[], int stack3[],
                              int n1, int n2, int n3)
{
  int sum1 = 0, sum2 = 0, sum3 = 0;

  // Finding the initial sum of stack1.
  for (int i=0; i < n1; i++)
      sum1 += stack1[i];

  // Finding the initial sum of stack2.
  for (int i=0; i < n2; i++)
      sum2 += stack2[i];

  // Finding the initial sum of stack3.
  for (int i=0; i < n3; i++)
      sum3 += stack3[i];

  // As given in question, first element is top
  // of stack..
  int top1 =0, top2 = 0, top3 = 0;
  int ans = 0;
  while (1)
  {
      // If any stack is empty
      if (top1 == n1 || top2 == n2 || top3 == n3)
         return 0;

      // If sum of all three stack are equal.
      if (sum1 == sum2 && sum2 == sum3)
         return sum1;

      // Finding the stack with maximum sum and
      // removing its top element.
      if (sum1 >= sum2 && sum1 >= sum3)
         sum1 -= stack1[top1++];
      else if (sum2 >= sum3 && sum2 >= sum3)
         sum2 -= stack2[top2++];
      else if (sum3 >= sum2 && sum3 >= sum1)
         sum3 -= stack3[top3++];
   }
}

// Driven Program
int main()
{
  int stack1[] = { 3, 2, 1, 1, 1 };
```

```
  int stack2[] = { 4, 3, 2 };
  int stack3[] = { 1, 1, 4, 1 };

  int n1 = sizeof(stack1)/sizeof(stack1[0]);
  int n2 = sizeof(stack2)/sizeof(stack2[0]);
  int n3 = sizeof(stack3)/sizeof(stack3[0]);

  cout << maxSum(stack1, stack2, stack3, n1, n2, n3) << endl;
  return 0;
}
```

Java

```
 // JAVA Code for Find maximum sum possible
// equal sum of three stacks
class GFG {

    // Returns maximum possible equal sum of three
    // stacks with removal of top elements allowed
    public static int maxSum(int stack1[], int stack2[],
                             int stack3[], int n1, int n2,
                                                   int n3)
    {
      int sum1 = 0, sum2 = 0, sum3 = 0;

      // Finding the initial sum of stack1.
      for (int i=0; i < n1; i++)
          sum1 += stack1[i];

      // Finding the initial sum of stack2.
      for (int i=0; i < n2; i++)
          sum2 += stack2[i];

      // Finding the initial sum of stack3.
      for (int i=0; i < n3; i++)
          sum3 += stack3[i];

      // As given in question, first element is top
      // of stack..
      int top1 =0, top2 = 0, top3 = 0;
      int ans = 0;
      while (true)
      {
          // If any stack is empty
          if (top1 == n1 || top2 == n2 || top3 == n3)
             return 0;

          // If sum of all three stack are equal.
```

```
          if (sum1 == sum2 && sum2 == sum3)
             return sum1;

          // Finding the stack with maximum sum and
          // removing its top element.
          if (sum1 >= sum2 && sum1 >= sum3)
             sum1 -= stack1[top1++];
          else if (sum2 >= sum3 && sum2 >= sum3)
             sum2 -= stack2[top2++];
          else if (sum3 >= sum2 && sum3 >= sum1)
             sum3 -= stack3[top3++];
       }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
          int stack1[] = { 3, 2, 1, 1, 1 };
          int stack2[] = { 4, 3, 2 };
          int stack3[] = { 1, 1, 4, 1 };

          int n1 = stack1.length;
          int n2 = stack2.length;
          int n3 = stack3.length;

          System.out.println(maxSum(stack1, stack2,
                                stack3, n1, n2, n3));
    }
  }
// This code is contributed by Arnav Kr. Mandal.
```

Python

```
 # Python program to calculate maximum sum with equal
# stack sum.
# Returns maximum possible equal sum of three stacks
# with removal of top elements allowed
def maxSum(stack1, stack2, stack3, n1, n2, n3):
    sum1, sum2, sum3 = 0, 0, 0

  # Finding the initial sum of stack1.
    for i in range(n1):
        sum1 += stack1[i]

  # Finding the initial sum of stack2.
    for i in range(n2):
        sum2 += stack2[i]
```

```
  # Finding the initial sum of stack3.
    for i in range(n3):
        sum3 += stack3[i]

  # As given in question, first element is top
  # of stack..
    top1, top2, top3 = 0, 0, 0
    ans = 0
    while (1):
      # If any stack is empty
      if (top1 == n1 or top2 == n2 or top3 == n3):
          return 0

      # If sum of all three stack are equal.
      if (sum1 == sum2 and sum2 == sum3):
          return sum1

      # Finding the stack with maximum sum and
      # removing its top element.
      if (sum1 >= sum2 and sum1 >= sum3):
          sum1 -= stack1[top1]
          top1=top1+1
      elif (sum2 >= sum3 and sum2 >= sum3):
          sum2 -= stack2[top2]
          top2=top2+1
      elif (sum3 >= sum2 and sum3 >= sum1):
          sum3 -= stack3[top3]
          top3=top3+1

# Driven Program
stack1 = [ 3, 2, 1, 1, 1 ]
stack2 = [ 4, 3, 2 ]
stack3 = [ 1, 1, 4, 1 ]

n1 = len(stack1)
n2 = len(stack2)
n3 = len(stack3)

print maxSum(stack1, stack2, stack3, n1, n2, n3)

#This code is contributed by Afzal Ansari
```

C#

```
 // C# Code for Find maximum sum with
// equal sum of three stacks
using System;
```

```
class GFG {

    // Returns maximum possible equal
    // sum of three stacks with removal
    // of top elements allowed
    public static int maxSum(int[] stack1,
                int[] stack2, int[] stack3,
                    int n1, int n2, int n3)
    {

        int sum1 = 0, sum2 = 0, sum3 = 0;

        // Finding the initial sum of
        // stack1.
        for (int i = 0; i < n1; i++)
            sum1 += stack1[i];

        // Finding the initial sum of
        // stack2.
        for (int i = 0; i < n2; i++)
            sum2 += stack2[i];

        // Finding the initial sum of
        // stack3.
        for (int i = 0; i < n3; i++)
            sum3 += stack3[i];

        // As given in question, first
        // element is top of stack..
        int top1 = 0, top2 = 0, top3 = 0;

        while (true) {

            // If any stack is empty
            if (top1 == n1 || top2 == n2
                            || top3 == n3)
                return 0;

            // If sum of all three stack
            // are equal.
            if (sum1 == sum2 && sum2 == sum3)
                return sum1;

            // Finding the stack with maximum
            // sum and removing its top element.
            if (sum1 >= sum2 && sum1 >= sum3)
                sum1 -= stack1[top1++];
            else if (sum2 >= sum3 && sum2 >= sum3)
```

```
                sum2 -= stack2[top2++];
            else if (sum3 >= sum2 && sum3 >= sum1)
                sum3 -= stack3[top3++];
        }
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int[] stack1 = { 3, 2, 1, 1, 1 };
        int[] stack2 = { 4, 3, 2 };
        int[] stack3 = { 1, 1, 4, 1 };

        int n1 = stack1.Length;
        int n2 = stack2.Length;
        int n3 = stack3.Length;

        Console.Write(maxSum(stack1, stack2,
                        stack3, n1, n2, n3));
    }
}

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

PHP

```
 <?php
// PHP program to calculate maximum
// sum with equal stack sum.

// Returns maximum possible
// equal sum of three stacks
// with removal of top elements
// allowed
function maxSum($stack1, $stack2, $stack3,
                            $n1, $n2, $n3)
{
    $sum1 = 0; $sum2 = 0; $sum3 = 0;

    // Finding the initial sum of stack1.
    for ($i = 0; $i < $n1; $i++)
        $sum1 += $stack1[$i];

    // Finding the initial sum of stack2.
    for ($i = 0; $i < $n2; $i++)
        $sum2 += $stack2[$i];

    // Finding the initial sum of stack3.
```

```
    for ($i = 0; $i < $n3; $i++)
        $sum3 += $stack3[$i];

    // As given in question,
    // first element is top
    // of stack..
    $top1 =0;
    $top2 = 0;
    $top3 = 0;
    $ans = 0;
    while (1)
    {

        // If any stack is empty
        if ($top1 == $n1 || $top2 == $n2 ||
                              $top3 == $n3)
            return 0;

        // If sum of all three stack are equal.
        if ($sum1 == $sum2 && $sum2 == $sum3)
            return $sum1;

        // Finding the stack with
        // maximum sum and
        // removing its top element.
        if ($sum1 >= $sum2 && $sum1 >= $sum3)
                $sum1 -= $stack1[$top1++];

        else if ($sum2 >= $sum3 && $sum2 >=$sum3)
                $sum2 -= $stack2[$top2++];

        else if ($sum3 >= $sum2 && $sum3 >= $sum1)
                $sum3 -= $stack3[$top3++];
    }
}

// Driver Code
$stack1 = array(3, 2, 1, 1, 1);
$stack2 = array(4, 3, 2);
$stack3 = array(1, 1, 4, 1);

$n1 = sizeof($stack1);
$n2 = sizeof($stack2);
$n3 = sizeof($stack3);
echo maxSum($stack1, $stack2,
            $stack3, $n1,
            $n2, $n3) ;
```

```
// This code is contributed by nitin mittal
?>
```

Output:

```
5
```

Time Complexity : O(n1 + n2 + n3) where n1, n2 and n3 are sizes of three stacks.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/find-maximum-sum-possible-equal-sum-three-stacks/

Chapter 27

Find minimum number of currency notes and values that sum to given amount

Find minimum number of currency notes and values that sum to given amount - GeeksforGeeks

Given an amount, find the minimum number of notes of different denominations that sum upto the given amount. Starting from the highest denomination note, try to accommodate as many notes possible for given amount.

We may assume that we have infinite supply of notes of values {2000, 500, 200, 100, 50, 20, 10, 5, 1}

Examples:

```
Input : 800
Output : Currency  Count
         500 : 1
         200 : 1
         100 : 1

Input : 2456
Output : Currency  Count
         2000 : 1
         200 : 2
         50 : 1
         5 : 1
         1 : 1
```

This problem is a simple variation of coin change problem. Here Greedy approach works as given system is canonical (Please refer this and this for details)

Below is the program implementation to find number of notes:
C++

```
 // C++ program to accept an amount
// and count number of notes
#include <bits/stdc++.h>
using namespace std;

// function to count and
// print currency notes
void countCurrency(int amount)
{
    int notes[9] = { 2000, 500, 200, 100,
                     50, 20, 10, 5, 1 };
    int noteCounter[9] = { 0 };

    // count notes using Greedy approach
    for (int i = 0; i < 9; i++) {
        if (amount >= notes[i]) {
            noteCounter[i] = amount / notes[i];
            amount = amount - noteCounter[i] * notes[i];
        }
    }

    // Print notes
    cout << "Currency Count ->" << endl;
    for (int i = 0; i < 9; i++) {
        if (noteCounter[i] != 0) {
            cout << notes[i] << " : "
                << noteCounter[i] << endl;
        }
    }
}

// Driver function
int main()
{
    int amount = 868;
    countCurrency(amount);
    return 0;
}
```

Python3

```
 # Python3 program to accept an amount
# and count number of notes
```

```
# Function to count and print
# currency notes
def countCurrency(amount):

    notes = [2000, 500, 200, 100,
               50, 20, 10, 5, 1]

    noteCounter = [0, 0, 0, 0, 0,
                     0, 0, 0, 0]

    print ("Currency Count -> ")

    for i, j in zip(notes, noteCounter):
        if amount >= i:
            j = amount // i
            amount = amount - j * i
            print (i ," : ", j)

# Driver code
amount = 868
countCurrency(amount)
```

Java

```
 // Java program to accept an amount
// and count number of notes
import java.util.*;
import java.lang.*;

public class GfG{

    // function to count and
    // print currency notes
    public static void countCurrency(int amount)
    {
        int[] notes = new int[]{ 2000, 500, 200, 100, 50, 20, 10, 5, 1 };
        int[] noteCounter = new int[9];

        // count notes using Greedy approach
        for (int i = 0; i < 9; i++) {
            if (amount >= notes[i]) {
                noteCounter[i] = amount / notes[i];
                amount = amount - noteCounter[i] * notes[i];
            }
        }

        // Print notes
        System.out.println("Currency Count ->");
```

```
        for (int i = 0; i < 9; i++) {
            if (noteCounter[i] != 0) {
                System.out.println(notes[i] + " : "
                    + noteCounter[i]);
            }
        }
    }

    // driver function
    public static void main(String argc[]){
        int amount = 868;
        countCurrency(amount);
    }

    /* This code is contributed by Sagar Shukla */
}
```

C#

```
 // C# program to accept an amount
// and count number of notes
using System;

public class GfG{

    // function to count and
    // print currency notes
    public static void countCurrency(int amount)
    {
        int[] notes = new int[]{ 2000, 500, 200, 100, 50, 20, 10, 5, 1 };
        int[] noteCounter = new int[9];

        // count notes using Greedy approach
        for (int i = 0; i < 9; i++) {
            if (amount >= notes[i]) {
                noteCounter[i] = amount / notes[i];
                amount = amount - noteCounter[i] * notes[i];
            }
        }

        // Print notes
        Console.WriteLine("Currency Count ->");
        for (int i = 0; i < 9; i++) {
            if (noteCounter[i] != 0) {
                Console.WriteLine(notes[i] + " : "
                    + noteCounter[i]);
            }
        }
```

```
    }

    // Driver function
    public static void Main(){
        int amount = 868;
        countCurrency(amount);
    }


}

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

PHP

```
 <?php
// PHP program to accept an amount
// and count number of notes

// function to count and
// prcurrency notes
function countCurrency($amount)
{
    $notes = array(2000, 500, 200, 100,
                   50, 20, 10, 5, 1);
    $noteCounter = array(0, 0, 0, 0, 0,
                         0, 0, 0, 0, 0);

    // count notes using
    // Greedy approach
    for ($i = 0; $i < 9; $i++)
    {
        if ($amount >= $notes[$i])
        {
            $noteCounter[$i] = intval($amount /
                                      $notes[$i]);
            $amount = $amount -
                      $noteCounter[$i] *
                      $notes[$i];
        }
    }
    // Print notes
    echo ("Currency Count ->"."\n");
    for ($i = 0; $i < 9; $i++)
    {
        if ($noteCounter[$i] != 0)
        {
            echo ($notes[$i] . " : " .
```

```
                    $noteCounter[$i] . "\n");
            }
        }
}

// Driver Code
$amount = 868;
countCurrency($amount);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
Currency  Count ->
500 : 1
200 : 1
100 : 1
50 : 1
10 : 1
5 : 1
1 : 3
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/find-number-currency-notes-sum-upto-given-amount/

Chapter 28

Find minimum time to finish all jobs with given constraints

Find minimum time to finish all jobs with given constraints - GeeksforGeeks

Given an array of jobs with different time requirements. There are K identical assignees available and we are also given how much time an assignee takes to do one unit of the job. Find the minimum time to finish all jobs with following constraints.

- An assignee can be assigned only contiguous jobs. For example, an assignee cannot be assigned jobs 1 and 3, but not 2.
- Two assignees cannot share (or co-assigned) a job, i.e., a job cannot be partially assigned to one assignee and partially to other.

Input :

```
K:     Number of assignees available.
T:     Time taken by an assignee to finish one unit
       of job
job[]: An array that represents time requirements of
       different jobs.
```

Examples :

```
Input:  k = 2, T = 5, job[] = {4, 5, 10}
Output: 50
The minimum time required to finish all the jobs is 50.
There are 2 assignees available. We get this time by
assigning {4, 5} to first assignee and {10} to second
```

assignee.

```
Input:  k = 4, T = 5, job[] = {10, 7, 8, 12, 6, 8}
Output: 75
We get this time by assigning {10} {7, 8} {12} and {6, 8}
```

We strongly recommend you to minimize your browser and try this yourself first.
The idea is to use Binary Search. Think if we have a function (say isPossible()) that tells us if it's possible to finish all jobs within a given time and number of available assignees. We can solve this problem by doing a binary search for the answer. If the middle point of binary search is not possible, then search in second half, else search in first half. Lower bound for Binary Search for minimum time can be set as 0. The upper bound can be obtained by adding all given job times.

Now how to implement isPossible()? This function can be implemented using Greedy Approach. Since we want to know if it is possible to finish all jobs within a given time, we traverse through all jobs and keep assigning jobs to current assignee one by one while a job can be assigned within the given time limit. When time taken by current assignee exceeds the given time, create a new assignee and start assigning jobs to it. If the number of assignees becomes more than k, then return false, else return true.

C++

```
 // C++ program to find minimum time to finish all jobs with
// given number of assignees
#include<bits/stdc++.h>
using namespace std;

// Utility function to get maximum element in job[0..n-1]
int getMax(int arr[], int n)
{
   int result = arr[0];
   for (int i=1; i<n; i++)
      if (arr[i] > result)
         result = arr[i];
    return result;
}

// Returns true if it is possible to finish jobs[] within
// given time 'time'
bool isPossible(int time, int K, int job[], int n)
{
    // cnt is count of current assignees required for jobs
    int cnt = 1;

    int curr_time = 0; //  time assigned to current assignee

    for (int i = 0; i < n;)
```

```
    {
        // If time assigned to current assignee exceeds max,
        // increment count of assignees.
        if (curr_time + job[i] > time) {
            curr_time = 0;
            cnt++;
        }
        else { // Else add time of job to current time and move
               // to next job.
            curr_time += job[i];
            i++;
        }
    }

    // Returns true if count is smaller than k
    return (cnt <= K);
}

// Returns minimum time required to finish given array of jobs
// k --> number of assignees
// T --> Time required by every assignee to finish 1 unit
// m --> Number of jobs
int findMinTime(int K, int T, int job[], int n)
{
    // Set start and end for binary search
    // end provides an upper limit on time
    int end = 0, start = 0;
    for (int i = 0; i < n; ++i)
        end += job[i];

    int ans = end; // Initialize answer

    // Find the job that takes maximum time
    int job_max = getMax(job, n);

    // Do binary search for minimum feasible time
    while (start <= end)
    {
        int mid = (start + end) / 2;

        // If it is possible to finish jobs in mid time
        if (mid >= job_max && isPossible(mid, K, job, n))
        {
            ans = min(ans, mid);  // Update answer
            end = mid - 1;
        }
        else
            start = mid + 1;
```

```
    }

    return (ans * T);
}

// Driver program
int main()
{
    int job[] =  {10, 7, 8, 12, 6, 8};
    int n = sizeof(job)/sizeof(job[0]);
    int k = 4, T = 5;
    cout << findMinTime(k, T, job, n) << endl;
    return 0;
}
```

Java

```
 // Java program to find minimum time
// to finish all jobs with given
// number of assignees

class GFG
{
    // Utility function to get
    // maximum element in job[0..n-1]
    static int getMax(int arr[], int n)
    {
    int result = arr[0];
    for (int i=1; i<n; i++)
        if (arr[i] > result)
            result = arr[i];
        return result;
    }

    // Returns true if it is possible to finish jobs[]
    // within given time 'time'
    static boolean isPossible(int time, int K,
                                  int job[], int n)
    {
        // cnt is count of current
        // assignees required for jobs
        int cnt = 1;

        // time assigned to current assignee
        int curr_time = 0;

        for (int i = 0; i < n;)
        {
```

```
            // If time assigned to current assignee
            // exceeds max, increment count of assignees.
            if (curr_time + job[i] > time) {
                curr_time = 0;
                cnt++;
            }

            // Else add time of job to current
            // time and move to next job.
            else
            {
                curr_time += job[i];
                i++;
            }
        }

        // Returns true if count
        // is smaller than k
        return (cnt <= K);
    }

    // Returns minimum time required to
    // finish given array of jobs
    // k --> number of assignees
    // T --> Time required by every assignee to finish 1 unit
    // m --> Number of jobs
    static int findMinTime(int K, int T, int job[], int n)
    {
        // Set start and end for binary search
        // end provides an upper limit on time
        int end = 0, start = 0;
        for (int i = 0; i < n; ++i)
            end += job[i];

        // Initialize answer
        int ans = end;

        // Find the job that takes maximum time
        int job_max = getMax(job, n);

        // Do binary search for
        // minimum feasible time
        while (start <= end)
        {
            int mid = (start + end) / 2;

            // If it is possible to finish jobs in mid time
            if (mid >= job_max && isPossible(mid, K, job, n))
```

```
            {
                // Update answer
                ans = Math.min(ans, mid);

                end = mid - 1;
            }

            else
                start = mid + 1;
        }

        return (ans * T);
    }

    // Driver program
    public static void main(String arg[])
    {
        int job[] = {10, 7, 8, 12, 6, 8};
        int n = job.length;
        int k = 4, T = 5;
        System.out.println(findMinTime(k, T, job, n));
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find minimum time
// to finish all jobs with given
// number of assignees
using System;

class GFG
{
    // Utility function to get
    // maximum element in job[0..n-1]
    static int getMax(int []arr, int n)
    {
      int result = arr[0];
      for (int i=1; i<n; i++)
        if (arr[i] > result)
            result = arr[i];
        return result;
    }

    // Returns true if it is possible to
    // finish jobs[] within given time 'time'
```

```
static bool isPossible(int time, int K,
                       int []job, int n)
{
    // cnt is count of current
    // assignees required for jobs
    int cnt = 1;

    // time assigned to current assignee
    int curr_time = 0;

    for (int i = 0; i < n;)
    {
        // If time assigned to current assignee
        // exceeds max, increment count of assignees.
        if (curr_time + job[i] > time) {
            curr_time = 0;
            cnt++;
        }

        // Else add time of job to current
        // time and move to next job.
        else
        {
            curr_time += job[i];
            i++;
        }
    }

    // Returns true if count
    // is smaller than k
    return (cnt <= K);
}

// Returns minimum time required to
// finish given array of jobs
// k --> number of assignees
// T --> Time required by every assignee to finish 1 unit
// m --> Number of jobs
static int findMinTime(int K, int T, int []job, int n)
{
    // Set start and end for binary search
    // end provides an upper limit on time
    int end = 0, start = 0;
    for (int i = 0; i < n; ++i)
        end += job[i];

    // Initialize answer
    int ans = end;
```

```
        // Find the job that takes maximum time
        int job_max = getMax(job, n);

        // Do binary search for
        // minimum feasible time
        while (start <= end)
        {
            int mid = (start + end) / 2;

            // If it is possible to finish jobs in mid time
            if (mid >= job_max && isPossible(mid, K, job, n))
            {
                // Update answer
                ans = Math.Min(ans, mid);

                end = mid - 1;
            }

            else
                start = mid + 1;
        }

        return (ans * T);
    }

    // Driver program
    public static void Main()
    {
        int []job = {10, 7, 8, 12, 6, 8};
        int n = job.Length;
        int k = 4, T = 5;
        Console.WriteLine(findMinTime(k, T, job, n));
    }
}

// This code is contributed by Sam007.
```

Output:

```
75
```

Thanks to Gaurav Ahirwar for suggesting above solution.

Improved By : Sam007

Source

https://www.geeksforgeeks.org/find-minimum-time-to-finish-all-jobs-with-given-constraints/

Chapter 29

Find smallest number with given number of digits and sum of digits

Find smallest number with given number of digits and sum of digits - GeeksforGeeks

How to find the smallest number with given digit sum **s** and number of digits **d**?

Examples :

```
Input  : s = 9, d = 2
Output : 18
There are many other possible numbers
like 45, 54, 90, etc with sum of digits
as 9 and number of digits as 2. The
smallest of them is 18.

Input  : s = 20, d = 3
Output : 299
```

A **Simple Solution** is to consider all m digit numbers and keep track of minimum number with digit sum as s. A close upper bound on time complexity of this solution is $O(10^m)$.

There is a **Greedy approach** to solve the problem. The idea is to one by one fill all digits from rightmost to leftmost (or from least significant digit to most significant).
We initially deduct 1 from sum s so that we have smallest digit at the end. After deducting 1, we apply greedy approach. We compare remaining sum with 9, if remaining sum is more than 9, we put 9 at the current position, else we put the remaining sum. Since we fill digits from right to left, we put the highest digits on the right side. Below is implementation of the idea.

C++

```
 // C++ program to find the smallest number that can be
// formed from given sum of digits and number of digits.
#include <iostream>
using namespace std;

// Prints the smallest possible number with digit sum 's'
// and 'm' number of digits.
void findSmallest(int m, int s)
{
    // If sum of digits is 0, then a number is possible
    // only if number of digits is 1.
    if (s == 0)
    {
        (m == 1)? cout << "Smallest number is " << 0
                : cout << "Not possible";
        return ;
    }

    // Sum greater than the maximum possible sum.
    if (s > 9*m)
    {
        cout << "Not possible";
        return ;
    }

    // Create an array to store digits of result
    int res[m];

    // deduct sum by one to account for cases later
    // (There must be 1 left for the most significant
    //  digit)
    s -= 1;

    // Fill last m-1 digits (from right to left)
    for (int i=m-1; i>0; i--)
    {
        // If sum is still greater than 9,
        // digit must be 9.
        if (s > 9)
        {
            res[i] = 9;
            s -= 9;
        }
        else
        {
            res[i] = s;
            s = 0;
        }
```

```
    }

    // Whatever is left should be the most significant
    // digit.
    res[0] = s + 1;  // The initially subtracted 1 is
                     // incorporated here.

    cout << "Smallest number is ";
    for (int i=0; i<m; i++)
        cout << res[i];
}

// Driver code
int main()
{
    int s = 9, m = 2;
    findSmallest(m, s);
    return 0;
}
```

Java

```
 // Java program to find the smallest number that can be
// formed from given sum of digits and number of digits

class GFG
{
    // Function to print the smallest possible number with digit sum 's'
    // and 'm' number of digits
    static void findSmallest(int m, int s)
    {
        // If sum of digits is 0, then a number is possible
        // only if number of digits is 1
        if (s == 0)
        {
            System.out.print(m == 1 ? "Smallest number is 0" : "Not possible");

            return ;
        }

        // Sum greater than the maximum possible sum
        if (s > 9*m)
        {
            System.out.println("Not possible");
            return ;
        }

        // Create an array to store digits of result
```

```
        int[] res = new int[m];

        // deduct sum by one to account for cases later
        // (There must be 1 left for the most significant
        //  digit)
        s -= 1;

        // Fill last m-1 digits (from right to left)
        for (int i=m-1; i>0; i--)
        {
            // If sum is still greater than 9,
            // digit must be 9
            if (s > 9)
            {
                res[i] = 9;
                s -= 9;
            }
            else
            {
                res[i] = s;
                s = 0;
            }
        }

        // Whatever is left should be the most significant
        // digit
        res[0] = s + 1;  // The initially subtracted 1 is
                         // incorporated here

        System.out.print("Smallest number is ");
        for (int i=0; i<m; i++)
            System.out.print(res[i]);
    }

    // driver program
    public static void main (String[] args)
    {
        int s = 9, m = 2;
        findSmallest(m, s);
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # Prints the smallest possible
# number with digit sum 's'
```

```
# and 'm' number of digits.

def findSmallest(m,s):

    # If sum of digits is 0,
    # then a number is possible
    # only if number of digits is 1.
    if (s == 0):

        if(m == 1) :
              print("Smallest number is 0")
        else :
              print("Not possible")
        return

    # Sum greater than the
    # maximum possible sum.
    if (s > 9*m):

        print("Not possible")
        return

    # Create an array to
    # store digits of result
    res=[0 for i in range(m+1)]

    # deduct sum by one to
    # account for cases later
    # (There must be 1 left
    # for the most significant
    #  digit)
    s -= 1

    # Fill last m-1 digits
    # (from right to left)
    for i in range(m-1,0,-1):

        # If sum is still greater than 9,
        # digit must be 9.
        if (s > 9):

            res[i] = 9
            s -= 9

        else:

            res[i] = s
            s = 0
```

```
    # Whatever is left should
    # be the most significant
    # digit.
    # The initially subtracted 1 is
    # incorporated here.
    res[0] = s + 1


    print("Smallest number is ",end="")
    for i in range(m):
        print(res[i],end="")


s = 9
m = 2
findSmallest(m, s)

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find the smallest
// number that can be formed from
// given sum of digits and number
// of digits
using System;

class GFG
{
    // Function to print the smallest
    // possible number with digit sum 's'
    // and 'm' number of digits
    static void findSmallest(int m, int s)
    {
        // If sum of digits is 0,
        // then a number is possible
        // only if number of digits is 1
        if (s == 0)
        {
            Console.Write(m == 1 ?
                          "Smallest number is 0" :
                                  "Not possible");

            return ;
        }
```

```
    // Sum greater than the
    // maximum possible sum
    if (s > 9 * m)
    {
        Console.Write("Not possible");
        return ;
    }

    // Create an array to
    // store digits of result
    int []res = new int[m];

    // deduct sum by one to account
    // for cases later (There must be
    // 1 left for the most significant
    // digit)
    s -= 1;

    // Fill last m-1 digits
    // (from right to left)
    for (int i = m - 1; i > 0; i--)
    {
        // If sum is still greater
        // than 9, digit must be 9
        if (s > 9)
        {
            res[i] = 9;
            s -= 9;
        }
        else
        {
            res[i] = s;
            s = 0;
        }
    }

    // Whatever is left should be
    // the most significant digit

    // The initially subtracted 1 is
    // incorporated here
    res[0] = s + 1;

    Console.Write("Smallest number is ");
    for (int i = 0; i < m; i++)
        Console.Write(res[i]);
}
```

```
    // Driver Code
    public static void Main ()
    {
        int s = 9, m = 2;
        findSmallest(m, s);
    }
}

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

PHP

```
 <?php
// PHP program to find the smallest
// number that can be formed from
// given sum of digits and number
// of digits.

// Prints the smallest possible
// number with digit sum 's'
// and 'm' number of digits.
function findSmallest($m, $s)
{
    // If sum of digits is 0, then
    // a number is possible only if
    // number of digits is 1.
    if ($s == 0)
    {
        if(($m == 1) == true)

        echo "Smallest number is " , 0;
        else
        echo "Not possible";
        return ;
    }

    // Sum greater than the
    // maximum possible sum.
    if ($s > 9 * $m)
    {
        echo "Not possible";
        return ;
    }

    // Create an array to store
    // digits of result int res[m];
    // deduct sum by one to account
    // for cases later (There must
```

```
    // be 1 left for the most
    // significant digit)
    $s -= 1;

    // Fill last m-1 digits
    // (from right to left)
    for ($i = $m - 1; $i > 0; $i--)
    {
        // If sum is still greater
        // than 9, digit must be 9.
        if ($s > 9)
        {
            $res[$i] = 9;
            $s -= 9;
        }
        else
        {
            $res[$i] = $s;
            $s = 0;
        }
    }

    // Whatever is left should be
    // the most significant digit.

    // The initially subtracted 1
    // is incorporated here.
    $res[0] = $s + 1;

    echo "Smallest number is ";
    for ($i = 0; $i < $m; $i++)
        echo $res[$i];
}

// Driver code
$s = 9; $m = 2;
findSmallest($m, $s);

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

Output :

```
Smallest number is 18
```

Time Complexity of this solution is O(m).

We will soon be discussing approach to find the largest possible number with given sum of digits and number of digits.

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/find-smallest-number-with-given-number-of-digits-and-digit-sum/

Chapter 30

Find the Largest Cube formed by Deleting minimum Digits from a number

Find the Largest Cube formed by Deleting minimum Digits from a number - GeeksforGeeks

Given a number n, the task is to find the largest perfect cube that can be formed by deleting minimum digits(possibly 0) from the number.
X is called a perfect cube if $X = Y^3$ for some Y.

Examples:

```
Input : 4125
Output : 125
Explanation
125 = 53. We can form 125 by deleting digit 4 from 4125

Input : 876
Output :8
Explanation
8 = 23. We can form 8 by deleting digits 7 and 6 from 876
```

We can generate cubes of all numbers till from 1 to $N^{1/3}$ (We don't consider 0 as 0 is not considered as a perfect cube). We iterate the cubes from largest to the smallest.
Now if we look at the number n given to us, then we know that this number contains only log(n) + 1 digits, thus we can efficiently approach the problem if we treat this number n as a string hereafter.
While iterating on the perfect cubes, we check if the perfect cube is a subsequence of the number n when its represented as a string.If this is the case then the deletions required for changing the number n to the current perfect cube is:

```
No of deleted digits = No of digits in number n -
                       Number of digits in current
                                    perfect cube
```

Since we want the largest cube number we traverse the array of preprocessed cubes in reverse order.

```
 /* C++ code to implement maximum perfect cube
   formed after deleting  minimum digits */
#include <bits/stdc++.h>
using namespace std;

// Returns vector of Pre Processed perfect cubes
vector<string> preProcess(long long int n)
{
    vector<string> preProcessedCubes;
    for (int i = 1; i * i * i <= n; i++) {
        long long int iThCube = i * i * i;

        // convert the cube to string and push into
        // preProcessedCubes vector
        string cubeString = to_string(iThCube);
        preProcessedCubes.push_back(cubeString);
    }
    return preProcessedCubes;
}

/* Utility function for findLargestCube().
   Returns the Largest cube number that can be formed */
string findLargestCubeUtil(string num,
                    vector<string> preProcessedCubes)
{
    // reverse the preProcessed cubes so that we
    // have the largest cube in the beginning
    // of the vector
    reverse(preProcessedCubes.begin(), preProcessedCubes.end());

    int totalCubes = preProcessedCubes.size();

    // iterate over all cubes
    for (int i = 0; i < totalCubes; i++) {
        string currCube = preProcessedCubes[i];

        int digitsInCube = currCube.length();
        int index = 0;
        int digitsInNumber = num.length();
        for (int j = 0; j < digitsInNumber; j++) {
```

```
            // check if the current digit of the cube
            // matches with that of the number num
            if (num[j] == currCube[index])
                index++;

            if (digitsInCube == index)
                return currCube;
        }
    }

    // if control reaches here, the its
    // not possible  to form a perfect cube
    return "Not Possible";
}

// wrapper for findLargestCubeUtil()
void findLargestCube(long long int n)
{
    // pre process perfect cubes
    vector<string> preProcessedCubes = preProcess(n);

    // convert number n to string
    string num = to_string(n);

    string ans = findLargestCubeUtil(num, preProcessedCubes);

    cout << "Largest Cube that can be formed from "
         << n << " is " << ans << endl;
}

// Driver Code
int main()
{
    long long int n;
    n = 4125;
    findLargestCube(n);

    n = 876;
    findLargestCube(n);

    return 0;
}
```

Output:

```
Largest Cube that can be formed from 4125 is 125
```

```
Largest Cube that can be formed from 876 is 8
```

Time Complexity of the above algorithm is $O(N^{1/3}\log(N)\ \log(N)$ is due to the fact that the number of digits in N are Log(N) + 1.

Source

https://www.geeksforgeeks.org/find-largest-cube-formed-deleting-minimum-digits-number/

Chapter 31

Find the Largest number with given number of digits and sum of digits

Find the Largest number with given number of digits and sum of digits - GeeksforGeeks

How to find the largest number with given digit sum **s** and number of digits **d**?

Examples:

```
Input  : s = 9, d = 2
Output : 90

Input  : s = 20, d = 3
Output : 992
```

A **Simple Solution** is to consider all m digit numbers and keep track of maximum number with digit sum as s. A close upper bound on time complexity of this solution is $O(10^m)$.

There is a **Greedy approach** to solve the problem. The idea is to one by one fill all digits from leftmost to rightmost (or from most significant digit to least significant).
We compare remaining sum with 9, if remaining sum is more than 9, we put 9 at the current position, else we put the remaining sum. Since we fill digits from left to right, we put the highest digits on the left side, hence get the largest number. Below is implementation of the idea.

C++

```
 // C++ program to find the largest number that can be
// formed from given sum of digits and number of digits.
#include <iostream>
```

```
using namespace std;

// Prints the smalles possible number with digit sum 's'
// and 'm' number of digits.
void findLargest(int m, int s)
{
    // If sum of digits is 0, then a nuber is possible
    // only if number of digits is 1.
    if (s == 0)
    {
        (m == 1)? cout << "Largest number is " << 0
                       : cout << "Not possible";
        return ;
    }

    // Sum greater than the maximum possible sum.
    if (s > 9*m)
    {
        cout << "Not possible";
        return ;
    }

    // Create an array to store digits of result
    int res[m];

    // Fill from most significant digit to least
    // significant digit.
    for (int i=0; i<m; i++)
    {
        // Fill 9 first to make the number largest
        if (s >= 9)
        {
            res[i] = 9;
            s -= 9;
        }

        // If remaining sum becomes less than 9, then
        // fill the remaining sum
        else
        {
            res[i] = s;
            s = 0;
        }
    }

    cout << "Largest number is ";
    for (int i=0; i<m; i++)
        cout << res[i];
```

```
}

// Driver code
int main()
{
    int s = 9, m = 2;
    findLargest(m, s);
    return 0;
}
```

Java

```
 // Java program to find the largest number that can be
// formed from given sum of digits and number of digits

class GFG
{
    // Function to print the largest possible number with digit sum 's'
    // and 'm' number of digits
    static void findLargest(int m, int s)
    {
        // If sum of digits is 0, then a number is possible
        // only if number of digits is 1
        if (s == 0)
        {
            System.out.print(m == 1 ? "Largest number is 0" : "Not possible");

            return ;
        }

        // Sum greater than the maximum possible sum
        if (s > 9*m)
        {
            System.out.println("Not possible");
            return ;
        }

        // Create an array to store digits of result
        int[] res = new int[m];

        // Fill from most significant digit to least
        // significant digit
        for (int i=0; i<m; i++)
        {
            // Fill 9 first to make the number largest
            if (s >= 9)
            {
                res[i] = 9;
```

```
                s -= 9;
            }

            // If remaining sum becomes less than 9, then
            // fill the remaining sum
            else
            {
                res[i] = s;
                s = 0;
            }
        }

        System.out.print("Largest number is ");
        for (int i=0; i<m; i++)
            System.out.print(res[i]);
    }

    // driver program
    public static void main (String[] args)
    {
        int s = 9, m = 2;
        findLargest(m, s);
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # Python 3 program to find
# the largest number that
# can be formed from given
# sum of digits and number
# of digits.


# Prints the smalles
# possible number with digit
# sum 's' and 'm' number of
# digits.
def findLargest( m, s) :

    # If sum of digits is 0,
    # then a nuber is possible
    # only if number of digits
    # is 1.
    if (s == 0) :
```

```
        if(m == 1) :
            print("Largest number is " , "0",end = "")
        else :
            print("Not possible",end = "")

        return

    # Sum greater than the
    # maximum possible sum.
    if (s > 9 * m) :
        print("Not possible",end = "")
        return

    # Create an array to
    # store digits of
    # result
    res = [0] * m

    # Fill from most significant
    # digit to least significant
    # digit.
    for i in range(0, m) :

        # Fill 9 first to make
        # the number largest
        if (s >= 9) :
            res[i] = 9
            s = s - 9

        # If remaining sum
        # becomes less than
        # 9, then fill the
        # remaining sum
        else :
            res[i] = s
            s = 0


    print( "Largest number is ",end = "")

    for i in range(0, m) :
        print(res[i],end = "")

# Driver code
s = 9
m = 2
findLargest(m, s)
```

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

C#

```
 // C# program to find the
// largest number that can
// be formed from given sum
// of digits and number of digits
using System;

class GFG
{

    // Function to print the
    // largest possible number
    // with digit sum 's' and
    // 'm' number of digits
    static void findLargest(int m, int s)
    {
        // If sum of digits is 0,
        // then a number is possible
        // only if number of digits is 1
        if (s == 0)
        {
            Console.Write(m == 1 ?
                   "Largest number is 0" :
                          "Not possible");

            return ;
        }

        // Sum greater than the
        // maximum possible sum
        if (s > 9 * m)
        {
            Console.WriteLine("Not possible");
            return ;
        }

        // Create an array to
        // store digits of result
        int []res = new int[m];

        // Fill from most significant
        // digit to least significant digit
        for (int i = 0; i < m; i++)
        {
            // Fill 9 first to make
```

```
            // the number largest
            if (s >= 9)
            {
                res[i] = 9;
                s -= 9;
            }

            // If remaining sum becomes
            // less than 9, then
            // fill the remaining sum
            else
            {
                res[i] = s;
                s = 0;
            }
        }

        Console.Write("Largest number is ");
        for (int i = 0; i < m; i++)
            Console.Write(res[i]);
    }

    // Driver Code
    static public void Main ()
    {
        int s = 9, m = 2;
        findLargest(m, s);
    }
}

// This code is Contributed by ajit
```

PHP

```
 <?php
// PHP program to find the largest
// number that can be formed from
// given sum of digits and number
// of digits.

// Prints the smalles possible
// number with digit sum 's'
// and 'm' number of digits.
function findLargest($m, $s)
{
    // If sum of digits is 0, then
    // a number is possible only if
    // number of digits is 1.
```

```
    if ($s == 0)
    {
        if(($m == 1) == true)
            echo "Largest number is " , 0;
        else
            echo "Not possible";
        return ;
    }

    // Sum greater than the
    // maximum possible sum.
    if ($s > 9 * $m)
    {
        echo "Not possible";
        return ;
    }

    // Create an array to store
    // digits of result Fill from
    // most significant digit to
    // least significant digit.
    for ($i = 0; $i < $m; $i++)
    {
        // Fill 9 first to make
        // the number largest
        if ($s >= 9)
        {
            $res[$i] = 9;
            $s -= 9;
        }

        // If remaining sum becomes
        // less than 9, then fill
        // the remaining sum
        else
        {
            $res[$i] = $s;
            $s = 0;
        }
    }

    echo "Largest number is ";
    for ($i = 0; $i < $m; $i++)
        echo $res[$i];
}

// Driver code
$s = 9; $m = 2;
```

```
findLargest($m, $s);

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

Output :

```
Largest number is 90
```

Time Complexity of this solution is O(m).

Improved By : jit_t

Source

https://www.geeksforgeeks.org/find-the-largest-number-with-given-number-of-digits-and-sum-of-digits/

Chapter 32

Find the minimum and maximum amount to buy all N candies

Find the minimum and maximum amount to buy all N candies - GeeksforGeeks

In a candy store there are N different types of candies available and the prices of all the N different types of candies are provided. There is also an attractive offer by candy store. We can buy a single candy from the store and get at-most K other candies (all are different types) for free.

1. Find minimum amount of money we have to spend to buy all the N different candies.
2. Find maximum amount of money we have to spend to buy all the N different candies.

In both the cases we must utilize the offer and get maximum possible candies back. If k or more candies are available, we must take k candies for every candy purchase. If less than k candies are available, we must take all candies for a candy purchase.

Examples:

```
Input :  price[] = {3, 2, 1, 4}
                k = 2
Output :  Min = 3, Max = 7
Since k is 2, if we buy one candy we can take
atmost two more for free.
So in the first case we buy the candy which
costs 1 and take candies worth 3 and 4 for
free, also you buy candy worth 2 as well.
So min cost = 1 + 2 = 3.
```

```
In the second case we buy the candy which
costs 4 and take candies worth 1 and 2 for
free, also We buy candy worth 3 as well.
So max cost = 3 + 4 = 7.
```

One important thing to note is, we must use the offer and get maximum candies back for every candy purchase. So if we want to minimize the money, we must buy candies of minimum cost and get candies of maximum costs for free. To maximize the money, we must do reverse. Below is algorithm based on this.

```
First Sort the price array.

For finding minimum amount :
  Start purchasing candies from starting
  and reduce k free candies from last with
  every single purchase.

For finding maximum amount :
   Start purchasing candies from the end
   and reduce k free candies from starting
   in every single purchase.
```

C++

```
 // C++ implementation to find the minimum
// and maximum amount
#include<bits/stdc++.h>
using namespace std;

// Function to find the minimum amount
// to buy all candies
int findMinimum(int arr[], int n, int k)
{
    int res = 0;
    for (int i=0; i<n ; i++)
    {
        // Buy current candy
        res += arr[i];

        // And take k candies for free
        // from the last
        n = n-k;
    }
    return res;
}
```

```
// Function to find the maximum amount
// to buy all candies
int findMaximum(int arr[], int n, int k)
{
    int res = 0, index = 0;

    for (int i=n-1; i>=index; i--)
    {
        // Buy candy with maximum amount
        res += arr[i];

        // And get k candies for free from
        // the starting
        index += k;
    }
    return res;
}

// Driver code
int main()
{
    int arr[] = {3, 2, 1, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    sort(arr, arr+n);

    cout << findMinimum(arr, n, k)<<" "
         << findMaximum(arr, n, k)<<endl;
    return 0;
}
```

Java

```
 // Java implementation to find the
// minimum and maximum amount
import java.util.*;

class GFG {

    // Function to find the minimum
    // amount to buy all candies
    static int findMinimum(int arr[], int n, int k)
    {
        int res = 0;
        for (int i = 0; i < n; i++)
        {
            // Buy current candy
            res += arr[i];
```

```
            // And take k candies for free
            // from the last
            n = n - k;
        }
        return res;
    }

    // Function to find the maximum
    // amount to buy all candies
    static int findMaximum(int arr[], int n, int k)
    {
        int res = 0, index = 0;

        for (int i = n - 1; i >= index; i--)
        {
            // Buy candy with maximum amount
            res += arr[i];

            // And get k candies for free from
            // the starting
            index += k;
        }
        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 3, 2, 1, 4 };
        int n = arr.length;
        int k = 2;
        Arrays.sort(arr);

        System.out.println(findMinimum(arr, n, k) +
                           " " + findMaximum(arr, n, k));
    }
}

// This code is contributed by prerna saini
```

Python3

```
 # Python implementation
# to find the minimum
# and maximum amount

# Function to find
```

```
# the minimum amount
# to buy all candies
def findMinimum(arr,n,k):

    res = 0
    i=0
    while(n):

        # Buy current candy
        res += arr[i]

        # And take k
        # candies for free
        # from the last
        n = n-k
        i+=1
    return res

# Function to find
# the maximum amount
# to buy all candies
def findMaximum(arr, n, k):

    res = 0
    index = 0
    i=n-1
    while(i>=index):

        # Buy candy with
        # maximum amount
        res += arr[i]

        # And get k candies
        # for free from
        # the starting
        index += k
        i -= 1

    return res

# Driver code

arr = [3, 2, 1, 4]
n = len(arr)
k = 2

arr.sort()
```

```
print(findMinimum(arr, n, k)," ",
    findMaximum(arr, n, k))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# implementation to find the
// minimum and maximum amount
using System;

public class GFG {

    // Function to find the minimum
    // amount to buy all candies
    static int findMinimum(int []arr,
                           int n, int k)
    {
        int res = 0;
        for (int i = 0; i < n; i++)
        {

            // Buy current candy
            res += arr[i];

            // And take k candies for
            // free from the last
            n = n - k;
        }

        return res;
    }

    // Function to find the maximum
    // amount to buy all candies
    static int findMaximum(int []arr,
                           int n, int k)
    {
        int res = 0, index = 0;

        for (int i = n - 1; i >= index; i--)
        {
            // Buy candy with maximum
            // amount
            res += arr[i];

            // And get k candies for free
```

```
            // from the starting
            index += k;
        }

        return res;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 3, 2, 1, 4 };
        int n = arr.Length;
        int k = 2;
        Array.Sort(arr);

        Console.WriteLine(
          findMinimum(arr, n, k) + " "
             + findMaximum(arr, n, k));
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP implementation to find the minimum
// and maximum amount

// Function to find the minimum amount
// to buy all candies
function findMinimum($arr, $n,$k)
{
    $res = 0;
    for ($i = 0; $i < $n ; $i++)
    {

        // Buy current candy
        $res += $arr[$i];

        // And take k candies for free
        // from the last
        $n = $n - $k;
    }
    return $res;
}

// Function to find the maximum amount
```

```
// to buy all candies
function findMaximum($arr, $n, $k)
{
    $res = 0;
    $index = 0;

    for ($i = $n - 1; $i >= $index; $i--)
    {

        // Buy candy with maximum amount
        $res += $arr[$i];

        // And get k candies
        // for free from
        // the starting
        $index += $k;
    }
    return $res;
}

    // Driver Code
    $arr = array(3, 2, 1, 4);
    $n = sizeof($arr);
    $k = 2;
    sort($arr); sort($arr,$n);

    echo findMinimum($arr, $n, $k)," "
            ,findMaximum($arr, $n, $k);
    return 0;

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

Output:

```
3 7
```

Time Complexity : O(n log n)

Improved By : Sam007, nitin mittal

Source

https://www.geeksforgeeks.org/find-minimum-maximum-amount-buy-n-candies/

Chapter 33

Fitting Shelves Problem

Fitting Shelves Problem - GeeksforGeeks

Given length of wall w and shelves of two lengths m and n, find the number of each type of shelf to be used and the remaining empty space in the optimal solution so that the empty space is minimum. The larger of the two shelves is cheaper so it is preferred. However cost is secondary and first priority is to minimize empty space on wall.

Examples:

```
Input : w = 24 m = 3 n = 5
Output : 3 3 0
We use three units of both shelves
and 0 space is left.
3 * 3 + 3 * 5 = 24
So empty space  = 24 - 24 = 0
Another solution could have been 8 0 0
but since the larger shelf of length 5
is cheaper the former will be the answer.

Input : w = 29 m = 3 n = 9
Output : 0 3 2
0 * 3 + 3 * 9 = 27
29 - 27 = 2
```

A simple and efficient approach will be to try all possible combinations of shelves that fit within the length of the wall.
To implement this approach along with the constraint that larger shelf costs less than the smaller one, starting from 0, we increase no of larger type shelves till they can be fit. For each case we calculate the empty space and finally store that value which minimizes the empty space. if empty space is same in two cases we prefer the one with more no of larger shelves. Below is its implementation.

C++

```
 // C++ program to find minimum space and units
// of two shelves to fill a wall.
#include<bits/stdc++.h>
using namespace std;

void minSpacePreferLarge(int wall, int m, int n)
{
    // initializing output variables
    int num_m = 0, num_n = 0, min_empty = wall;

    // p and q are no of shelves of length m and n
    // rem is the empty space
    int p = 0, q = 0, rem;

    while (wall >= n)
    {
        //place one more shelf of length n
        q += 1;
        wall = wall - n;

        // place as many shelves of length m
        // in the remaining part
        p = wall / m;
        rem = wall % m;

        // update output variablse if curr
        // min_empty <= overall empty
        if (rem <= min_empty)
        {
            num_m = p;
            num_n = q;
            min_empty = rem;
        }
    }

    cout << num_m << " " << num_n << " "
        << min_empty << endl;
}

// Driver code
int main()
{
    int wall = 24, m = 3, n = 5;
    minSpacePreferLarge(wall, m, n);
    return 0;
}
```

Java

```
 // Java program to count all rotation
// divisible by 4.

public class GFG
{
    static void minSpacePreferLarge(int wall, int m, int n)
    {
        // initializing output variables
        int num_m = 0, num_n = 0, min_empty = wall;

        // p and q are no of shelves of length m and n
        // rem is the empty space
        int p = 0, q = 0, rem;

        while (wall >= n)
        {
            // place one more shelf of length n
            q += 1;
            wall = wall - n;

            // place as many shelves of length m
            // in the remaining part
            p = wall / m;
            rem = wall % m;

            // update output variablse if curr
            // min_empty <= overall empty
            if (rem <= min_empty)
            {
                num_m = p;
                num_n = q;
                min_empty = rem;
            }
        }
        System.out.println(num_m + " " + num_n + " " + min_empty);
    }

    public static void main(String[] args)
    {
        int wall = 24, m = 3, n = 5;
        minSpacePreferLarge(wall, m, n);
    }
}

// This code is contributed by Saket Kumar
```

Python

```
def minSpacePreferLarge(w, m, n):

    # initialize result variables
    num_m = 0
    num_n = 0
    rem = w

    # p and q are no of shelves of length m &
    # n respectively. r is the remainder uncovered
    # wall length
    p = 0
    q = 0
    r = 0
    while (w >= n):
        q += 1
        w -= n
        p = w / m
        r = w % m
        if (r <= rem):
            num_m = p
            num_n = q
            rem = r
    print( str(int(num_m)) + " " + str(num_n) + " " + str(rem))

# Driver code
w = 24
m = 3
n = 5
minSpacePreferLarge(w, m, n)
```

Output:

```
3 3 0
```

References: Sumologic Internship question

Source

https://www.geeksforgeeks.org/fitting-shelves-problem/

Chapter 34

Fractional Knapsack Problem

Fractional Knapsack Problem - GeeksforGeeks

Given weights and values of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or don't take it.

```
Input:
  Items as (value, weight) pairs
  arr[] = {{60, 10}, {100, 20}, {120, 30}}
  Knapsack Capacity, W = 50;
Output:
  Maximum possible value = 220
  by taking items of weight 20 and 30 kg
```

In **Fractional Knapsack**, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.

```
Input :
   Same as above
Output :
   Maximum possible value = 240
   By taking full items of 10 kg, 20 kg and
   2/3rd of last item of 30 kg
```

A **brute-force solution** would be to try all possible subset with all different fraction but that will be too much time taking.

An **efficient solution** is to use Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio.

Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.

A simple code with our own comparison function can be written as follows, please see sort function more closely, the third argument to sort function is our comparison function which sorts the item according to value/weight ratio in non-decreasing order.
After sorting we need to loop over these items and add them in our knapsack satisfying above-mentioned criteria.

```
 // C/C++ program to solve fractional Knapsack Problem
#include <bits/stdc++.h>
using namespace std;

// Structure for an item which stores weight and corresponding
// value of Item
struct Item
{
    int value, weight;

    // Constructor
    Item(int value, int weight) : value(value), weight(weight)
    {}
};

// Comparison function to sort Item according to val/weight ratio
bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

// Main greedy function to solve problem
double fractionalKnapsack(int W, struct Item arr[], int n)
{
    //    sorting Item on basis of ratio
    sort(arr, arr + n, cmp);

    //    Uncomment to see new order of Items with their ratio
    /*
    for (int i = 0; i < n; i++)
    {
        cout << arr[i].value << "  " << arr[i].weight << " : "
             << ((double)arr[i].value / arr[i].weight) << endl;
    }
    */

    int curWeight = 0;  // Current weight in knapsack
```

```
    double finalvalue = 0.0; // Result (value in Knapsack)

    // Looping through all Items
    for (int i = 0; i < n; i++)
    {
        // If adding Item won't overflow, add it completely
        if (curWeight + arr[i].weight <= W)
        {
            curWeight += arr[i].weight;
            finalvalue += arr[i].value;
        }

        // If we can't add current Item, add fractional part of it
        else
        {
            int remain = W - curWeight;
            finalvalue += arr[i].value * ((double) remain / arr[i].weight);
            break;
        }
    }

    // Returning final value
    return finalvalue;
}

// driver program to test above function
int main()
{
    int W = 50;   //    Weight of knapsack
    Item arr[] = {{60, 10}, {100, 20}, {120, 30}};

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

    cout << "Maximum value we can obtain = "
         << fractionalKnapsack(W, arr, n);
    return 0;
}
```

Output :

```
Maximum value in Knapsack = 240
```

As main time taking step is sorting, the whole problem can be solved in O(n log n) only.
This article is contributed by Utkarsh Trivedi.

Source

https://www.geeksforgeeks.org/fractional-knapsack-problem/

Chapter 35

Graph Coloring Set 2 (Greedy Algorithm)

Graph Coloring Set 2 (Greedy Algorithm) - GeeksforGeeks

We introduced graph coloring and applications in previous post. As discussed in the previous post, graph coloring is widely used. Unfortunately, there is no efficient algorithm available for coloring a graph with minimum number of colors as the problem is a known NP Complete problem. There are approximate algorithms to solve the problem though. Following is the basic Greedy Algorithm to assign colors. It doesn't guarantee to use minimum colors, but it guarantees an upper bound on the number of colors. The basic algorithm never uses more than d+1 colors where d is the maximum degree of a vertex in the given graph.

Basic Greedy Coloring Algorithm:

> **1.** Color first vertex with first color.
> **2.** Do following for remaining V-1 vertices.
> ..... **a)** Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Following are C++ and Java implementations of the above Greedy Algorithm.

C++

```
 // A C++ program to implement greedy algorithm for graph coloring
#include <iostream>
#include <list>
using namespace std;

// A class that represents an undirected graph
class Graph
```

```
{
    int V;    // No. of vertices
    list<int> *adj;    // A dynamic array of adjacency lists
public:
    // Constructor and destructor
    Graph(int V)   { this->V = V; adj = new list<int>[V]; }
    ~Graph()       { delete [] adj; }

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

    // Prints greedy coloring of the vertices
    void greedyColoring();
};

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
    adj[w].push_back(v);  // Note: the graph is undirected
}

// Assigns colors (starting from 0) to all vertices and prints
// the assignment of colors
void Graph::greedyColoring()
{
    int result[V];

    // Assign the first color to first vertex
    result[0]  = 0;

    // Initialize remaining V-1 vertices as unassigned
    for (int u = 1; u < V; u++)
        result[u] = -1;  // no color is assigned to u

    // A temporary array to store the available colors. True
    // value of available[cr] would mean that the color cr is
    // assigned to one of its adjacent vertices
    bool available[V];
    for (int cr = 0; cr < V; cr++)
        available[cr] = false;

    // Assign colors to remaining V-1 vertices
    for (int u = 1; u < V; u++)
    {
        // Process all adjacent vertices and flag their colors
        // as unavailable
        list<int>::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
```

```
            if (result[*i] != -1)
                available[result[*i]] = true;

        // Find the first available color
        int cr;
        for (cr = 0; cr < V; cr++)
            if (available[cr] == false)
                break;

        result[u] = cr; // Assign the found color

        // Reset the values back to false for the next iteration
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = false;
    }

    // print the result
    for (int u = 0; u < V; u++)
        cout << "Vertex " << u << " --->  Color "
             << result[u] << endl;
}

// Driver program to test above function
int main()
{
    Graph g1(5);
    g1.addEdge(0, 1);
    g1.addEdge(0, 2);
    g1.addEdge(1, 2);
    g1.addEdge(1, 3);
    g1.addEdge(2, 3);
    g1.addEdge(3, 4);
    cout << "Coloring of graph 1 \n";
    g1.greedyColoring();

    Graph g2(5);
    g2.addEdge(0, 1);
    g2.addEdge(0, 2);
    g2.addEdge(1, 2);
    g2.addEdge(1, 4);
    g2.addEdge(2, 4);
    g2.addEdge(4, 3);
    cout << "\nColoring of graph 2 \n";
    g2.greedyColoring();

    return 0;
}
```

Java

```
 // A Java program to implement greedy algorithm for graph coloring
import java.io.*;
import java.util.*;
import java.util.LinkedList;

// This class represents an undirected graph using adjacency list
class Graph
{
    private int V;   // No. of vertices
    private LinkedList<Integer> adj[]; //Adjacency List

    //Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    //Function to add an edge into the graph
    void addEdge(int v,int w)
    {
        adj[v].add(w);
        adj[w].add(v); //Graph is undirected
    }

    // Assigns colors (starting from 0) to all vertices and
    // prints the assignment of colors
    void greedyColoring()
    {
        int result[] = new int[V];

        // Initialize all vertices as unassigned
        Arrays.fill(result, -1);

        // Assign the first color to first vertex
        result[0]  = 0;

        // A temporary array to store the available colors. False
        // value of available[cr] would mean that the color cr is
        // assigned to one of its adjacent vertices
        boolean available[] = new boolean[V];

        // Initially, all colors are available
        Arrays.fill(available, true);
```

```
    // Assign colors to remaining V-1 vertices
    for (int u = 1; u < V; u++)
    {
        // Process all adjacent vertices and flag their colors
        // as unavailable
        Iterator<Integer> it = adj[u].iterator() ;
        while (it.hasNext())
        {
            int i = it.next();
            if (result[i] != -1)
                available[result[i]] = false;
        }

        // Find the first available color
        int cr;
        for (cr = 0; cr < V; cr++){
            if (available[cr])
                break;
        }

        result[u] = cr; // Assign the found color

        // Reset the values back to true for the next iteration
        Arrays.fill(available, true);
    }

    // print the result
    for (int u = 0; u < V; u++)
        System.out.println("Vertex " + u + " --->  Color "
                           + result[u]);
}

// Driver method
public static void main(String args[])
{
    Graph g1 = new Graph(5);
    g1.addEdge(0, 1);
    g1.addEdge(0, 2);
    g1.addEdge(1, 2);
    g1.addEdge(1, 3);
    g1.addEdge(2, 3);
    g1.addEdge(3, 4);
    System.out.println("Coloring of graph 1");
    g1.greedyColoring();

    System.out.println();
    Graph g2 = new Graph(5);
```

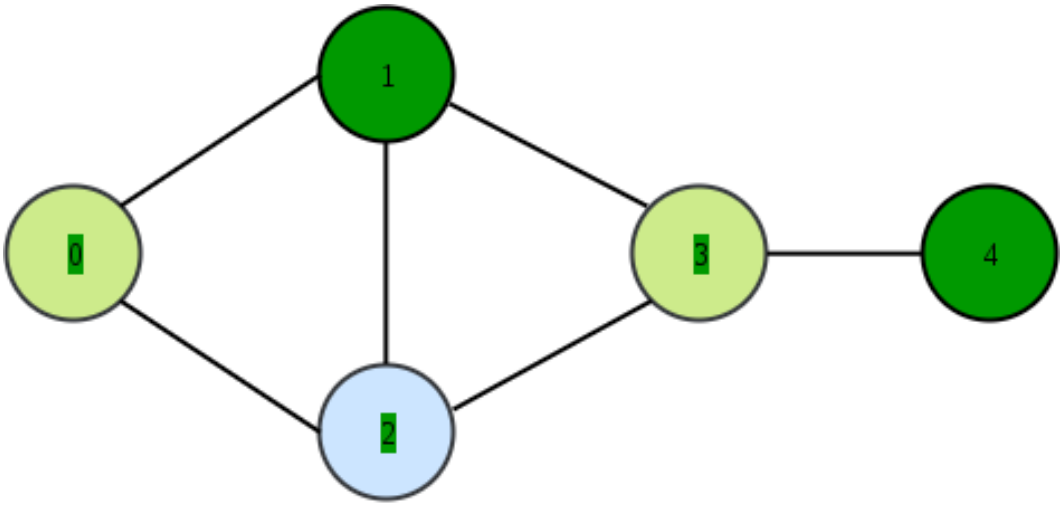
```
        g2.addEdge(0, 1);
        g2.addEdge(0, 2);
        g2.addEdge(1, 2);
        g2.addEdge(1, 4);
        g2.addEdge(2, 4);
        g2.addEdge(4, 3);
        System.out.println("Coloring of graph 2 ");
        g2.greedyColoring();
    }
}
// This code is contributed by Aakash Hasija
```

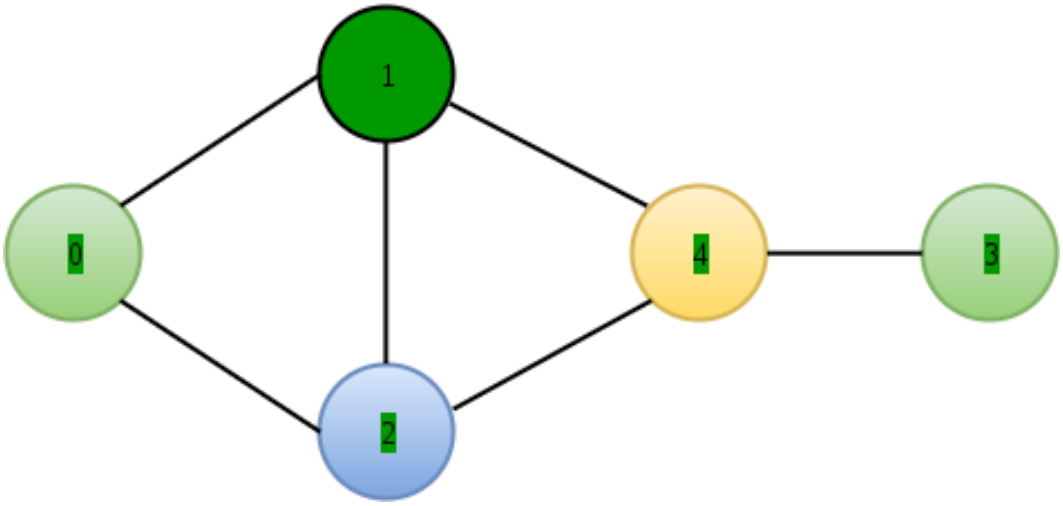
Output:

```
Coloring of graph 1
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 1

Coloring of graph 2
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 3
```

Time Complexity: O(V^2 + E) in worst case.

Analysis of Basic Algorithm
The above algorithm doesn't always use minimum number of colors. Also, the number of colors used sometime depend on the order in which vertices are processed. For example, consider the following two graphs. Note that in graph on right side, vertices 3 and 4 are swapped. If we consider the vertices 0, 1, 2, 3, 4 in left graph, we can color the graph using 3 colors. But if we consider the vertices 0, 1, 2, 3, 4 in right graph, we need 4 colors.





So the order in which the vertices are picked is important. Many people have suggested different ways to find an ordering that work better than the basic algorithm on average. The most common is Welsh–Powell Algorithm which considers vertices in descending order of degrees.

How does the basic algorithm guarantee an upper bound of d+1?
Here d is the maximum degree in the given graph. Since d is maximum degree, a vertex cannot be attached to more than d vertices. When we color a vertex, at most d colors could have already been used by its adjacent. To color this vertex, we need to pick the smallest numbered color that is not used by the adjacent vertices. If colors are numbered like 1, 2, ..., then the value of such smallest number must be between 1 to d+1 (Note that d numbers are already picked by adjacent vertices).

This can also be proved using induction. See thisvideo lecture for proof.

We will soon be discussing some interesting facts about chromatic number and graph coloring.

Improved By : ChamanJhinga

Source

https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/

Chapter 36

Greedy Algorithm for Egyptian Fraction

Greedy Algorithm for Egyptian Fraction - GeeksforGeeks

Every positive fraction can be represented as sum of unique unit fractions. A fraction is unit fraction if numerator is 1 and denominator is a positive integer, for example 1/3 is a unit fraction. Such a representation is called Egyptian Fraction as it was used by ancient Egyptians.

Following are few examples:

```
Egyptian Fraction Representation of 2/3 is 1/2 + 1/6
Egyptian Fraction Representation of 6/14 is 1/3 + 1/11 + 1/231
Egyptian Fraction Representation of 12/13 is 1/2 + 1/3 + 1/12 + 1/156
```

We can generate Egyptian Fractions using Greedy Algorithm. For a given number of the form 'nr/dr' where dr > nr, first find the greatest possible unit fraction, then recur for the remaining part. For example, consider 6/14, we first find ceiling of 14/6, i.e., 3. So the first unit fraction becomes 1/3, then recur for (6/14 – 1/3) i.e., 4/42.

Below is implementation of above idea.

C++

```
 // C++ program to print a fraction in Egyptian Form using Greedy
// Algorithm
#include <iostream>
using namespace std;

void printEgyptian(int nr, int dr)
{
    // If either numerator or denominator is 0
```

```
    if (dr == 0 || nr == 0)
        return;

    // If numerator divides denominator, then simple division
    // makes the fraction in 1/n form
    if (dr%nr == 0)
    {
        cout << "1/" << dr/nr;
        return;
    }

    // If denominator divides numerator, then the given number
    // is not fraction
    if (nr%dr == 0)
    {
        cout << nr/dr;
        return;
    }

    // If numerator is more than denominator
    if (nr > dr)
    {
        cout << nr/dr << " + ";
        printEgyptian(nr%dr, dr);
        return;
    }

    // We reach here dr > nr and dr%nr is non-zero
    // Find ceiling of dr/nr and print it as first
    // fraction
    int n = dr/nr + 1;
    cout << "1/" << n << " + ";

    // Recur for remaining part
    printEgyptian(nr*n-dr, dr*n);
 }

// Driver Program
int main()
{
    int nr = 6, dr = 14;
    cout << "Egyptian Fraction Representation of "
         << nr << "/" << dr << " is\n ";
    printEgyptian(nr, dr);
    return 0;
}
```

Output:

```
Egyptian Fraction Representation of 6/14 is
 1/3 + 1/11 + 1/231
```

The Greedy algorithm works because a fraction is always reduced to a form where denominator is greater than numerator and numerator doesn't divide denominator. For such reduced forms, the highlighted recursive call is made for reduced numerator. So the recursive calls keep on reducing the numerator till it reaches 1.

References:
http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fractions/egyptian.html

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.

Improved By : DhruvitJakasaniya

Source

https://www.geeksforgeeks.org/greedy-algorithm-egyptian-fraction/

Chapter 37

Greedy Algorithm to find Minimum number of Coins

Greedy Algorithm to find Minimum number of Coins - GeeksforGeeks

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.
```

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;
}
```

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.

Source

https://www.geeksforgeeks.org/greedy-algorithm-to-find-minimum-number-of-coins/

Chapter 38

Huffman Coding Greedy Algo-3

Huffman Coding Greedy Algo-3 - GeeksforGeeks

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.

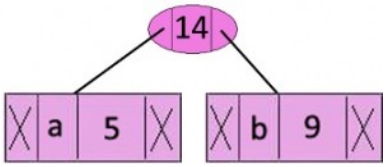
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

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.

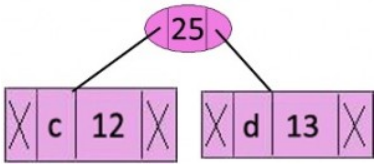
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.



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

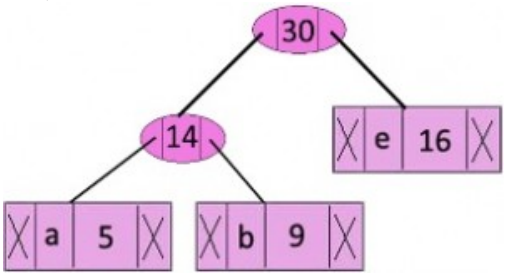


Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes.

character	Frequency

```
Internal Node          14
       e               16
Internal Node          25
       f               45
```

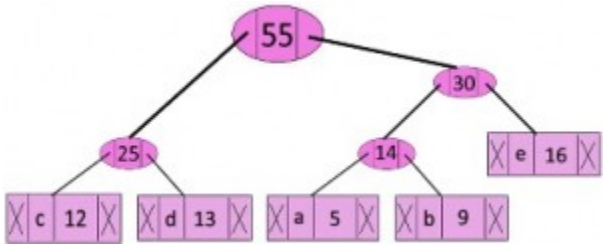
Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30



Now min heap contains 3 nodes.

```
character          Frequency
Internal Node         25
Internal Node         30
      f               45
```

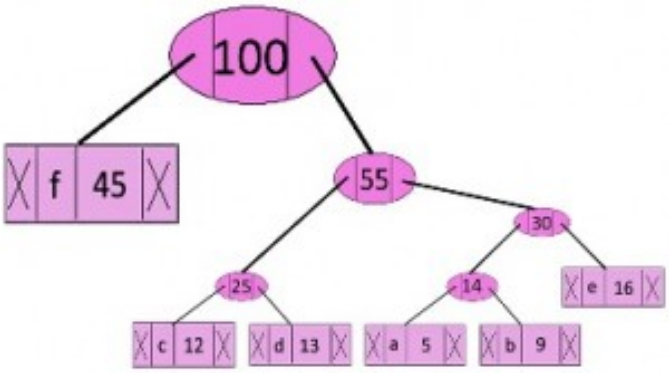
Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55



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

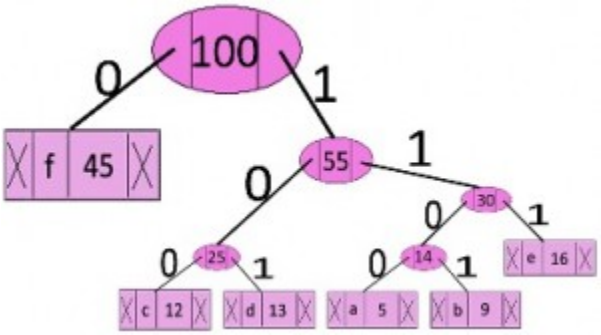


Now min heap contains only one node.

```
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

```
 // 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 {

    // One of the input characters
    char data;

    // Frequency of the character
    unsigned freq;

    // Left and right child of this node
    struct MinHeapNode *left, *right;
};

// A Min Heap:  Collection of
// min heap (or Hufmman tree) nodes
struct MinHeap {

    // Current size of min heap
    unsigned size;

    // capacity of min heap
    unsigned capacity;

    // Attay of minheap node pointers
    struct MinHeapNode** array;
};

// 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));

    // current size is 0
    minHeap->size = 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 funvtion 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;
}
```

C++ using STL

```
 // C++ program for Huffman Coding
#include <bits/stdc++.h>
using namespace std;

// A Huffman tree node
struct MinHeapNode {

    // One of the input characters
    char data;

    // Frequency of the character
    unsigned freq;

    // Left and right child
    MinHeapNode *left, *right;

    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
```

Java

```
 import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.Comparator;

// node class is the basic structure
// of each node present in the huffman - tree.
class HuffmanNode {

    int data;
    char c;

    HuffmanNode left;
    HuffmanNode right;
}

// comparator class helps to compare the node
// on the basis of one of its attribute.
```

```
// Here we will be compared
// on the basis of data values of the nodes.
class MyComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y)
    {

        return x.data - y.data;
    }
}

public class Huffman {

    // recursive function to print the
    // huffman-code through the tree traversal.
    // Here s is the huffman - code generated.
    public static void printCode(HuffmanNode root, String s)
    {

        // base case; if the left and right are null
        // then its a leaf node and we print
        // the code s generated by traversing the tree.
        if (root.left
                == null
            && root.right
                   == null
            && Character.isLetter(root.c)) {

            // c is the character in the node
            System.out.println(root.c + ":" + s);

            return;
        }

        // if we go to left then add "0" to the code.
        // if we go to the right add"1" to the code.

        // recursive calls for left and
        // right sub-tree of the generated tree.
        printCode(root.left, s + "0");
        printCode(root.right, s + "1");
    }

    // main function
    public static void main(String[] args)
    {

        Scanner s = new Scanner(System.in);
```

```
// number of characters.
int n = 6;
char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
int[] charfreq = { 5, 9, 12, 13, 16, 45 };

// creating a priority queue q.
// makes a min-priority queue(min-heap).
PriorityQueue<HuffmanNode> q
    = new PriorityQueue<HuffmanNode>(n, new MyComparator());

for (int i = 0; i < n; i++) {

    // creating a huffman node object
    // and adding it to the priority-queue.
    HuffmanNode hn = new HuffmanNode();

    hn.c = charArray[i];
    hn.data = charfreq[i];

    hn.left = null;
    hn.right = null;

    // add functions adds
    // the huffman node to the queue.
    q.add(hn);
}

// create a root node
HuffmanNode root = null;

// Here we will extract the two minimum value
// from the heap each time until
// its size reduces to 1, extract until
// all the nodes are extracted.
while (q.size() > 1) {

    // first min extract.
    HuffmanNode x = q.peek();
    q.poll();

    // second min extarct.
    HuffmanNode y = q.peek();
    q.poll();

    // new node f which is equal
    HuffmanNode f = new HuffmanNode();

    // to the sum of the frequency of the two nodes
```

```
            // assigning values to the f node.
            f.data = x.data + y.data;
            f.c = '-';

            // first extracted node as left child.
            f.left = x;

            // second extracted node as the right child.
            f.right = y;

            // marking the f node as the root node.
            root = f;

            // add this node to the priority-queue.
            q.add(f);
        }

        // print the codes by traversing the tree
        printCode(root, "");
    }
}

// This code is contributed by Kunwar Desh Deepak Singh
```

```
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 calles 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.

Improved By : kddeepak

Source

https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

Chapter 39

Huffman Decoding

Huffman Decoding - GeeksforGeeks

We have discussed Huffman Encoding in a previous post. In this post decoding is discussed.

Examples:

```
Input Data : AAAAAABCCCCCCDDEEEEE
Frequencies : A: 6, B: 1, C: 6, D: 2, E: 5
Encoded Data :
00000000000011001010101010111111101010101O
Huffman Tree: '#' is the special character used
              for internal nodes as character field
              is not needed for internal nodes.
               #(20)
             /       \
        #(12)         #(8)
     /      \        /    \
    A(6)     C(6) E(5)     #(3)
                          /    \
                        B(1)    D(2)
Code of 'A' is '00', code of 'C' is '01', ..
Decoded Data : AAAAAABCCCCCCDDEEEEE
```

```
Input Data : GeeksforGeeks
Character With there Frequencies
e 10, f 1100, g 011, k 00, o 010, r 1101, s 111
Encoded Huffman data :
011101000111110001011010111010000111
Decoded Huffman Data
geeksforgeeks
```

To decode the encoded data we require the Huffman tree. We iterate through the binary encoded data. To find character corresponding to current bits, we use following simple steps.

1. We start from root and do following until a leaf is found.
2. If current bit is 0, we move to left node of the tree.
3. If the bit is 1, we move to right node of the tree.
4. If during traversal, we encounter a leaf node, we print character of that particular leaf node and then again continue the iteration of the encoded data starting from step 1.

The below code takes a string as input, it encodes it and save in a variable encodedString. Then it decodes it and print the original string.

The below code performs full Huffman Encoding and Decoding of a given input data.

```
 // C++ program to encode and decode a string using
// Huffman Coding.
#include <bits/stdc++.h>
#define MAX_TREE_HT 256
using namespace std;

// to map each character its huffman value
map<char, string> codes;

// to store the frequency of character of the input data
map<char, int> freq;

// A Huffman tree node
struct MinHeapNode
{
    char data;             // One of the input characters
    int freq;             // Frequency of the character
    MinHeapNode *left, *right; // Left and right child

    MinHeapNode(char data, int freq)
    {
        left = right = NULL;
        this->data = data;
        this->freq = freq;
    }
};

// utility function for the priority queue
struct compare
{
    bool operator()(MinHeapNode* l, MinHeapNode* r)
    {
```

```
        return (l->freq > r->freq);
    }
};

// utility function to print characters along with
// there huffman value
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");
}

// utility function to store characters along with
// there huffman value in a hash table, here we
// have C++ STL map
void storeCodes(struct MinHeapNode* root, string str)
{
    if (root==NULL)
        return;
    if (root->data != '$')
        codes[root->data]=str;
    storeCodes(root->left, str + "0");
    storeCodes(root->right, str + "1");
}

// STL priority queue to store heap tree, with respect
// to their heap root node value
priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;

// function to build the Huffman tree and store it
// in minHeap
void HuffmanCodes(int size)
{
    struct MinHeapNode *left, *right, *top;
    for (map<char, int>::iterator v=freq.begin(); v!=freq.end(); v++)
        minHeap.push(new MinHeapNode(v->first, v->second));
    while (minHeap.size() != 1)
    {
        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();
        top = new MinHeapNode('$', left->freq + right->freq);
        top->left = left;
```

```
        top->right = right;
        minHeap.push(top);
    }
    storeCodes(minHeap.top(), "");
}

// utility function to store map each character with its
// frequency in input string
void calcFreq(string str, int n)
{
    for (int i=0; i<str.size(); i++)
        freq[str[i]]++;
}

// function iterates through the encoded string s
// if s[i]=='1' then move to node->right
// if s[i]=='0' then move to node->left
// if leaf node append the node->data to our output string
string decode_file(struct MinHeapNode* root, string s)
{
    string ans = "";
    struct MinHeapNode* curr = root;
    for (int i=0;i<s.size();i++)
    {
        if (s[i] == '0')
           curr = curr->left;
        else
           curr = curr->right;

        // reached leaf node
        if (curr->left==NULL and curr->right==NULL)
        {
            ans += curr->data;
            curr = root;
        }
    }
    // cout<<ans<<endl;
    return ans+'\0';
}

// Driver program to test above functions
int main()
{
    string str = "geeksforgeeks";
    string encodedString, decodedString;
    calcFreq(str, str.length());
    HuffmanCodes(str.length());
    cout << "Character With there Frequencies:\n";
```

```
    for (auto v=codes.begin(); v!=codes.end(); v++)
        cout << v->first <<' ' << v->second << endl;

    for (auto i: str)
        encodedString+=codes[i];

    cout << "\nEncoded Huffman data:\n" << encodedString << endl;

    decodedString = decode_file(minHeap.top(), encodedString);
    cout << "\nDecoded Huffman Data:\n" << decodedString << endl;
    return 0;
}
```

Output:

```
Character With there Frequencies
e 10
f 1100
g 011
k 00
o 010
r 1101
s 111

Encoded Huffman data
01110100011111000101101011101000111

Decoded Huffman Data
geeksforgeeks
```

Comparing Input file size and Output file size:
Comparing the input file size and the Huffman encoded output file. We can calculate the size of the output data in a simple way. Lets say our input is a string "geeksforgeeks" and is stored in a file input.txt.
Input File Size:

```
Input: "geeksforgeeks"
Total number of character i.e. input length: 13
Size: 13 character occurrences * 8 bits = 104 bits or 13 bytes.
```

Output File Size:

```
Input: "geeksforgeeks"
```

```
------------------------------------------------
Character |  Frequency |  Binary Huffman Value |
------------------------------------------------
   e      |      4     |          10           |
   f      |      1     |          1100         |
   g      |      2     |          011          |
   k      |      2     |          00           |
   o      |      1     |          010          |
   r      |      1     |          1101         |
   s      |      2     |          111          |
------------------------------------------------

So to calculate output size:
e: 4 occurrences * 2 bits = 8 bits
f: 1 occurrence  * 4 bits = 4 bits
g: 2 occurrences * 3 bits = 6 bits
k: 2 occurrences * 2 bits = 4 bits
o: 1 occurrence  * 3 bits = 3 bits
r: 1 occurrence  * 4 bits = 4 bits
s: 2 occurrences * 3 bits = 6 bits

Total Sum: 35 bits approx 5 bytes
```

Hence, we could see that after encoding the data we have saved a large amount of data. The above method can also help us to determine the value of N i.e. the length of the encoded data.

Source

https://www.geeksforgeeks.org/huffman-decoding/

Chapter 40

Job Scheduling with two jobs allowed at a time

Job Scheduling with two jobs allowed at a time - GeeksforGeeks

We are given N jobs, and their starting and ending times. We can do two jobs simultaneously at a particular moment. If one job ends at the same moment some other show starts then we can't do them. We need to check if it is possible to complete all the jobs or not.

Examples:

```
Input :  Start and End times of Jobs
         1 2
         2 3
         4 5
Output : Yes
By the time third job starts, both jobs
are finished. So we can schedule third
job.

Input : Start and End times of Jobs
        1 5
        2 4
        2 6
        1 7
Output : No
All 4 jobs needs to be scheduled at time
3 which is not possible.
```

We first sort the jobs according to their starting time. Then we start two jobs simultaneously and check if the starting time of third job and so on is greater than the ending time of and of the previous two jobs.
The C++ implementation the above idea is given below.

```
 // CPP program to check if all jobs can be scheduled
// if two jobs are allowed at a time.
#include <bits/stdc++.h>
using namespace std;

bool checkJobs(int startin[], int endin[], int n)
{
    // making a pair of starting and ending time of job
    vector<pair<int, int> > a;
    for (int i = 0; i < n; i++)
        a.push_back(make_pair(startin[i], endin[i]));

    // sorting according to starting time of job
    sort(a.begin(), a.end());

    // starting first and second job simultaneously
    long long tv1 = a[0].second, tv2 = a[1].second;

    for (int i = 2; i < n; i++) {

        // Checking if starting time of next new job
        // is greater than ending time of currently
        // scheduled first job
        if (a[i].first >= tv1)
        {
            tv2 = tv1;
            tv1 = a[i].second;
        }

        // Checking if starting time of next new job
        // is greater than ending time of ocurrently
        // scheduled second job
        else if (a[i].first >= tv2)
            tv2 = a[i].second;

        else
            return false;
    }
    return true;
}

// Driver code
int main()
{
    int startin[] = { 1, 2, 4 }; // starting time of jobs
    int endin[] = { 2, 3, 5 }; // ending times of jobs
    int n = sizeof(startin) / sizeof(startin[0]);
    cout << checkJobs(startin, endin, n);
```

```
    return 0;
}
```

Output:

```
1
```

An alternate solution is to find maximum number of jobs that needs to be scheduled at any time. If this count is more than 2, return false. Else return true.

Source

https://www.geeksforgeeks.org/job-scheduling-two-jobs-allowed-time/

Chapter 41

Job Selection Problem – Loss Minimization Strategy Set 2

Job Selection Problem - Loss Minimization Strategy Set 2 - GeeksforGeeks

We have discussed one loss minimization strategy before: Job Sequencing Problem – Loss Minimization. In this article, we will look at another strategy that applies to a slightly different problem.

We are given a sequence of N goods of production numbered from 1 to N. Each good has a volume denoted by (Vi). The constraint is that once a good has been completed its volume starts decaying at a fixed percentage (P) per day. All goods decay at the same rate and further each good take one day to complete.

We are required to find the order in which the goods should be produced so that overall volume of goods is maximized.

Example-1:

```
Input: 4, 2, 151, 15, 1, 52, 12 and P = 10%
Output: 222.503
```

Solution: In the optimum sequence of jobs, the total volume of goods left at the end of all jobs is 222.503

Example-2:

```
Input: 3, 1, 41, 52, 15, 4, 1, 63, 12 and P = 20%
Output: 145.742
```

Solution: In the optimum sequence of jobs the total volume of goods left at the end of all jobs is 145.72

Explanation –
Since this is an optimization problem, we can try to solve this problem by using a greedy algorithm. On each day we make a selection from among the goods that are yet to be produced. Thus all we need is a local selection criteria or heuristic, which when applied to select the jobs will give us the optimum result.

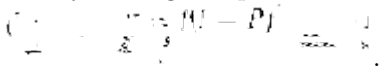
Instead of trying to maximize the volume, we can also try to minimize the losses. Since the total volume that can be obtained from all goods is also constant, if we minimize the losses we are guaranteed to get the optimum answer.

Now consider any good having volume V
Loss after Day 1: PV
Loss after Day 2: PV + P(1-P)V or V(2P-P^2)
Loss after Day 3: V(2P-P^2) + P(1 – 2P + P^2)V or V(3P – 3P^2 + P^3)

As the day increases the losses too increase. So the trick would be to ensure that the goods are not kept idle after production. Further, since we are required to produce at least one job per day, we should perform low volume jobs, and then perform the high volume jobs. This strategy works due to two factors.

1. High Volume goods are not kept idle after production.
2. As the volume decreases the loss per day too decreases, so for low volume goods the losses become negligible after a few days.

So in order to obtain the optimum solution we produce the larger volume goods later on. For the first day select the good with least volume and produce it. Remove the produced good from the list of goods. For the next day repeat the same. Keep repeating while there are goods left to be produced.

When calculating the total volume at the end of production, keep in mind the the good produced on day i, will have  times its volume left. Evidently, the good produced on day N (last day) will have its volume intact since  .

Algorithm –

```
Step 1: Add all the goods to a min-heap
Step 2: Repeat following steps while Queue is not empty
        Extract the good  at the head of the heap
        Print the good
        Remove the good from the heap
        [END OF LOOP]
Step 4: End
```

Complexity –
We perform exactly N push() and pop() operations each of which takes log (N) time. Hence time complexity is O(N * log(N)).

Below is the Cpp implementation of the solution.

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

void optimum_sequence_jobs(vector<int>& V, double P)
{
    int j = 1, N = V.size() - 1;
    double result = 0;

    // Create a min-heap (priority queue)
    priority_queue<int, vector<int>, greater<int> > Queue;

    // Add all goods to the the Queue
    for (int i = 1; i <= N; i++)
        Queue.push(V[i]);

    // Pop Goods from Queue as long as it is not empty
    while (!Queue.empty()) {

        // Print the good
        cout << Queue.top() << " ";

        // Add the Queue to the vector
        // so that total voulme can be calculated
        V[j++] = Queue.top();
        Queue.pop();
    }

    // Calclulating volume of goods left when all
    // are produced. Move from right to left of
    // sequence multiplying each volume by
    // increasing powers of 1 - P starting from 0
    for (int i = N; i >= 1; i--)
        result += pow((1 - P), N - i) * V[i];

    // Print result
    cout << endl << result << endl;
}

// Driver code
int main()
{
    // For implementation simplicity days are numbered
    // from 1 to N. Hence 1 based indexing is used
    vector<int> V{ -1, 3, 5, 4, 1, 2, 7, 6, 8, 9, 10 };

    // 10% loss per day
```

```
    double P = 0.10;

    optimum_sequence_jobs(V, P);

    return 0;
}
```

Output –

```
1 2 3 4 5 6 7 8 9 10
41.3811
```

Source

https://www.geeksforgeeks.org/job-selection-problem-loss-minimization-strategy-set-2/

Chapter 42

Job Sequencing Problem Set 1 (Greedy Algorithm)

Job Sequencing Problem Set 1 (Greedy Algorithm) - GeeksforGeeks

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
```

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 Algorithmproblem. 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 reqquired
void printJobScheduling(Job arr[], int n)
{
    // Sort all jobs according to decreasing order of prfit
    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 jobsn";
    printJobScheduling(arr, n);
    return 0;
}
```

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-engineerin
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.

Source

https://www.geeksforgeeks.org/job-sequencing-problem-set-1-greedy-algorithm/

Chapter 43

Job Sequencing Problem Set 2 (Using Disjoint Set)

Job Sequencing Problem Set 2 (Using Disjoint Set) - GeeksforGeeks

Given a set of n jobs where each job i has a deadline di >=1 and profit pi>=0. Only one job can be scheduled at a time. Each job takes 1 unit of time to complete. We earn the profit if and only if the job is completed by its deadline. The task is to find the subset of jobs that maximizes profit.

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
```

A greedy solution of time complexity $O(n^2)$ is already discussed here. Below is the simple Greedy 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
    - 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 costly operation in the Greedy solution is to assign a free slot for a job. We were traversing each and every slot for a job and assigning the greatest possible time slot(<deadline) which was available.

What does greatest time slot means?
Suppose that a job J1 has a deadline of time t = 5. We assign the greatest time slot which is free and less than the deadline i.e 4-5 for this job. Now another job J2 with deadline of 5 comes in, so the time slot allotted will be 3-4 since 4-5 has already been allotted to job J1.

Why to assign greatest time slot(free) to a job?
Now we assign the greatest possible time slot since if we assign a time slot even lesser than the available one than there might be some other job which will miss its deadline.
Example:
J1 with deadline d1 = 5, profit 40
J2 with deadline d2 = 1, profit 20
Suppose that for job J1 we assigned time slot of 0-1. Now job J2 cannot be performed since we will perform Job J1 during that time slot.

Using Disjoint Set for Job Sequencing
All time slots are individual sets initially. We first find the maximum deadline of all jobs. Let the max deadline be m. We create m+1 individual sets. If a job is assigned a time slot of t where t => 0, then the job is scheduled during [t-1, t]. So a set with value X represents the time slot [X-1, X].
We need to keep track of the greatest time slot available which can be allotted to a given job having deadline. We use the parent array of Disjoint Set Data structures for this purpose. The root of the tree is always the latest available slot. If for a deadline d, there is no slot available, then root would be 0. Below are detailed steps.

Initialize Disjoint Set: Creates initial disjoint sets.

```
// m is maximum deadline of a job
parent = new int[m + 1];

// Every node is a parent of itself
for (int i = 0; i   m; i++)
    parent[i] = i;
```

Find : Finds the latest time slot available.

```
// Returns the maximum available time slot
find(s)
{
    // Base case
    if (s == parent[s])
       return s;

    // Recursive call with path compression
    return parent[s] = find(parent[s]);
}
```

Union :

```
 Merges two sets.
// Makes u as parent of v.
union(u, v)
{
   // update the greatest available
   // free slot to u
   parent[v] = u;
}
```

How come find returns the latest available time slot?
Initially all time slots are individual slots. So the time slot returned is always maximum. When we assign a time slot 't' to a job, we do union of 't' with 't-1' in a way that 't-1' becomes parent of 't'. To do this we call union(t-1, t). This means that all future queries for time slot t would now return the latest time slot available for set represented by t-1.

Implementation :
The following is C++ implementation of above algorithm.

C++

```
 // C++ Program to find the maximum profit job sequence
// from a given array of jobs with deadlines and profits
#include<bits/stdc++.h>
using namespace std;

// A structure to represent various attributes of a Job
struct Job
{
    // Each job has id, deadline and profit
    char id;
    int deadLine, profit;
};

// A Simple Disjoint Set Data Structure
```

```
struct DisjointSet
{
    int *parent;

    // Constructor
    DisjointSet(int n)
    {
        parent = new int[n+1];

        // Every node is a parent of itself
        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }

    // Path Compression
    int find(int s)
    {
        /* Make the parent of the nodes in the path
           from u--> parent[u] point to parent[u] */
        if (s == parent[s])
            return s;
        return parent[s] = find(parent[s]);
    }

    // Makes u as parent of v.
    void merge(int u, int v)
    {
        //update the greatest available
        //free slot to u
        parent[v] = u;
    }
};

// Used to sort in descending order on the basis
// of profit for each job
bool cmp(Job a, Job b)
{
    return (a.profit > b.profit);
}

// Functions returns the maximum deadline from the set
// of jobs
int findMaxDeadline(struct Job arr[], int n)
{
    int ans = INT_MIN;
    for (int i = 0; i < n; i++)
        ans = max(ans, arr[i].deadLine);
    return ans;
```

```
}

int printJobScheduling(Job arr[], int n)
{
    // Sort Jobs in descending order on the basis
    // of their profit
    sort(arr, arr + n, cmp);

    // Find the maximum deadline among all jobs and
    // create a disjoint set data structure with
    // maxDeadline disjoint sets initially.
    int maxDeadline = findMaxDeadline(arr, n);
    DisjointSet ds(maxDeadline);

    // Traverse through all the jobs
    for (int i = 0; i < n; i++)
    {
        // Find the maximum available free slot for
        // this job (corresponding to its deadline)
        int availableSlot = ds.find(arr[i].deadLine);

        // If maximum available free slot is greater
        // than 0, then free slot available
        if (availableSlot > 0)
        {
            // This slot is taken by this job 'i'
            // so we need to update the greatest
            // free slot. Note that, in merge, we
            // make first parameter as parent of
            // second parameter. So future queries
            // for availableSlot will return maximum
            // available slot in set of
            // "availableSlot - 1"
            ds.merge(ds.find(availableSlot - 1),
                             availableSlot);

            cout << arr[i].id << " ";
        }
    }
}

// Driver program to test above function
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 jobs need to be "
         << "executed for maximum profit\n";
    printJobScheduling(arr, n);
    return 0;
    return 0;
}
```

Java

```
 // Java program to find the maximum profit job sequence
// from a given array of jobs with deadlines and profits
import java.util.*;

// A Simple Disjoint Set Data Structure
class DisjointSet
{
    int parent[];

    // Constructor
    DisjointSet(int n)
    {
        parent = new int[n + 1];

        // Every node is a parent of itself
        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }

    // Path Compression
    int find(int s)
    {
        /* Make the parent of the nodes in the path
           from u--> parent[u] point to parent[u] */
        if (s == parent[s])
            return s;
        return parent[s] = find(parent[s]);
    }

    // Makes u as parent of v.
    void merge(int u, int v)
    {
        //update the greatest available
        //free slot to u
        parent[v] = u;
    }
}

class Job implements Comparator<Job>
```

```
{
    // Each job has a unique-id, profit and deadline
    char id;
    int deadline, profit;

    // Constructors
    public Job() { }
    public Job(char id,int deadline,int profit)
    {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }

    // Returns the maximum deadline from the set of jobs
    public static int findMaxDeadline(ArrayList<Job> arr)
    {
        int ans = Integer.MIN_VALUE;
        for (Job temp : arr)
            ans = Math.max(temp.deadline, ans);
        return ans;
    }

    // Prints optimal job sequence
    public static void printJobScheduling(ArrayList<Job> arr)
    {
        // Sort Jobs in descending order on the basis
        // of their profit
        Collections.sort(arr, new Job());

        // Find the maximum deadline among all jobs and
        // create a disjoint set data structure with
        // maxDeadline disjoint sets initially.
        int maxDeadline = findMaxDeadline(arr);
        DisjointSet dsu = new DisjointSet(maxDeadline);

        // Traverse through all the jobs
        for (Job temp : arr)
        {
            // Find the maximum available free slot for
            // this job (corresponding to its deadline)
            int availableSlot = dsu.find(temp.deadline);


            // If maximum available free slot is greater
            // than 0, then free slot available
            if (availableSlot > 0)
            {
```

```
                // This slot is taken by this job 'i'
                // so we need to update the greatest free
                // slot. Note that, in merge, we make
                // first parameter as parent of second
                // parameter.  So future queries for
                // availableSlot will return maximum slot
                // from set of "availableSlot - 1"
                dsu.merge(dsu.find(availableSlot - 1),
                                  availableSlot);
                System.out.print(temp.id + " ");
            }
        }
        System.out.println();
    }

    // Used to sort in descending order on the basis
    // of profit for each job
    public int compare(Job j1, Job j2)
    {
        return j1.profit > j2.profit? -1: 1;
    }
}

// Driver code
class Main
{
    public static void main(String args[])
    {
        ArrayList<Job> arr=new ArrayList<Job>();
        arr.add(new Job('a',2,100));
        arr.add(new Job('b',1,19));
        arr.add(new Job('c',2,27));
        arr.add(new Job('d',1,25));
        arr.add(new Job('e',3,15));
        System.out.println("Following jobs need to be "+
                           "executed for maximum profit");
        Job.printJobScheduling(arr);
    }
}
```

Output:

```
Following jobs need to be executed for maximum profit
a c e
```

Source

https://www.geeksforgeeks.org/job-sequencing-using-disjoint-set-union/

Chapter 44

Job Sequencing Problem – Loss Minimization

Job Sequencing Problem - Loss Minimization - GeeksforGeeks

We are given N jobs numbered 1 to N. For each activity, let Ti denotes the number of days required to complete the job. For each day of delay before starting to work for job i, a loss of Li is incurred.
We are required to find a sequence to complete the jobs so that overall loss is minimized. We can only work on one job at a time.

If multiple such solutions are possible, then we are required to give the lexicographically least permutation (i.e earliest in dictionary order).

Examples:

```
Input : L = {3, 1, 2, 4} and
        T = {4, 1000, 2, 5}
Output : 3, 4, 1, 2
Explanation: We should first complete
job 3, then jobs 4, 1, 2 respectively.

Input : L = {1, 2, 3, 5, 6}
        T = {2, 4, 1, 3, 2}
Output : 3, 5, 4, 1, 2
Explanation: We should complete jobs
3, 5, 4, 1 and then 2 in this order.
```

Let us consider two extreme cases and we shall deduce the general case solution from them.

All jobs take same time to finish, i.e Ti = k for all i. Since all jobs take same time to finish we should first select jobs which have large Loss (Li). We should select jobs which have the

highest losses and finish them as early as possible.
Thus this is a greedy algorithm. Sort the jobs in descending order based on Li only.

All jobs have the same penalty. Since all jobs have the same penalty we will do those jobs first which will take less amount of time to finish. This will minimize the total delay, and hence also the total loss incurred.
This is also a greedy algorithm. Sort the jobs in ascending order based on Ti. Or we can also sort in descending order of 1/Ti.

Source

https://www.geeksforgeeks.org/job-sequencing-problem-loss-minimization/

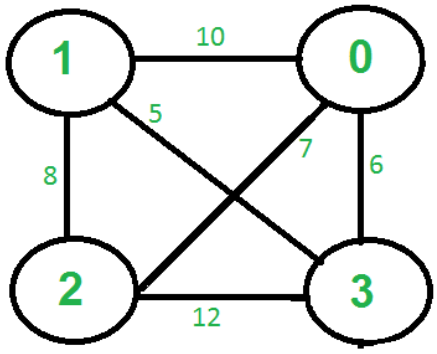
Chapter 45

K Centers Problem Set 1 (Greedy Approximate Algorithm)

K Centers Problem Set 1 (Greedy Approximate Algorithm) - GeeksforGeeks

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.



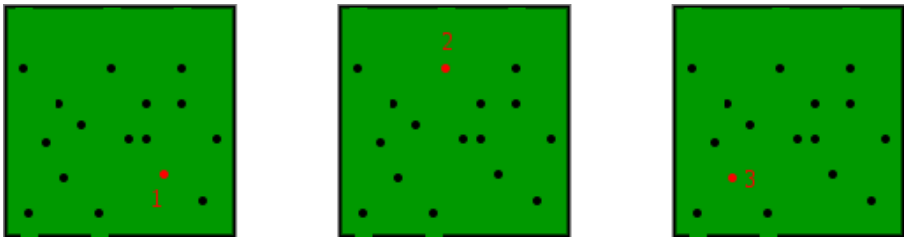
k = 2

The two ATMs should be placed in cities 2 and 3. The maximum distance of a city from an ATM becomes 6 in this optimal placement (We can not get the maximum distance less than 7)

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 that 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 c1, c2, c3, … ci 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
Min[dist(p, c1), dist(p, c2), dist(p, c3), …. dist(p, ci)]



Example (k = 3 in the above shown Graph)
a) Let the first arbitrarily picked vertex be 0.

b) The next vertex is 1 because 1 is the farthest vertex from 0.

c) 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 distanced from 2 to already considered centers
Min[dist(2, 0), dist(2, 1)] = Min[7, 8] = 7

Minimum of all distanced from 3 to already considered centers
Min[dist(3, 0), dist(3, 1)] = 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*OPT.

The proof can be done using contradiction.

a) Assume that the distance from the furthest point to all centers is $> 2 \cdot$ OPT.

b) This means that distances between all centers are also $> 2 \cdot$ OPT.

c) We have k + 1 points with distances $> 2 \cdot$ OPT between every pair.

d) Each point has a center of the optimal solution with distance <= OPT to it.

e) There exists a pair of points with the same center X in the optimal solution (pigeonhole principle: k optimal centers, k+1 points)

f) The distance between them is at most $2 \cdot$ 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

Source

https://www.geeksforgeeks.org/k-centers-problem-set-1-greedy-approximate-algorithm/

Chapter 46

Kruskal's Algorithm (Simple Implementation for Adjacency Matrix)

Kruskal's Algorithm (Simple Implementation for Adjacency Matrix) - GeeksforGeeks

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.

We have discussed one implementation of Kruskal's algorithm in previous post. In this post, a simpler implementation for adjacency matrix is discussed.

```
 // Simple C++ implementation for Kruskal's
// algorithm
#include <bits/stdc++.h>
using namespace std;

#define V 5
int parent[V];

// Find set of vertex i
int find(int i)
{
    while (parent[i] != i)
        i = parent[i];
    return i;
```

```
}

// Does union of i and j. It returns
// false if i and j are already in same
// set.
void union1(int i, int j)
{
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

// Finds MST using Kruskal's algorithm
void kruskalMST(int cost[][V])
{
    int mincost = 0; // Cost of min MST.

    // Initialize sets of disjoint sets.
    for (int i = 0; i < V; i++)
        parent[i] = i;

    // Include minimum weight edges one by one
    int edge_count = 0;
    while (edge_count < V - 1) {
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (find(i) != find(j) && cost[i][j] < min) {
                    min = cost[i][j];
                    a = i;
                    b = j;
                }
            }
        }

        union1(a, b);
        printf("Edge %d:(%d, %d) cost:%d \n",
               edge_count++, a, b, min);
        mincost += min;
    }
    printf("\n Minimum cost= %d \n", mincost);
}

// 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 cost[][V] = {
        { INT_MAX, 2, INT_MAX, 6, INT_MAX },
        { 2, INT_MAX, 3, 8, 5 },
        { INT_MAX, 3, INT_MAX, INT_MAX, 7 },
        { 6, 8, INT_MAX, INT_MAX, 9 },
        { INT_MAX, 5, 7, 9, INT_MAX },
    };

    // Print the solution
    kruskalMST(cost);

    return 0;
}
```

Output:

```
Edge 0:(0, 1) cost:2
Edge 1:(1, 2) cost:3
Edge 2:(1, 4) cost:5
Edge 3:(0, 3) cost:6

 Minimum cost= 16
```

Note that the above solution is not efficient. The idea is to provide a simple implementation for adjacency matrix representations. Please see below for efficient implementations.
Kruskal's Minimum Spanning Tree Algorithm Greedy Algo-2
Kruskal's Minimum Spanning Tree using STL in C++

Source

https://www.geeksforgeeks.org/kruskals-algorithm-simple-implementation-for-adjacency-matrix/

Chapter 47

Kruskal's Minimum Spanning Tree Algorithm Greedy Algo-2

Kruskal's Minimum Spanning Tree Algorithm Greedy Algo-2 - GeeksforGeeks

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.

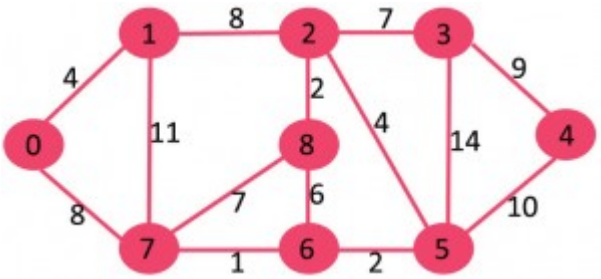
What are the applications of Minimum Spanning Tree?
See thisfor applications of MST.

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.

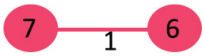


The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

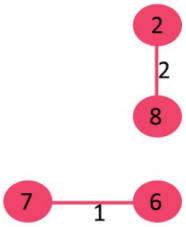
```
After sorting:
Weight   Src    Dest
1          7      6
2          8      2
2          6      5
4          0      1
4          2      5
6          8      6
7          2      3
7          7      8
8          0      7
8          1      2
9          3      4
10         5      4
11         1      7
14         3      5
```

Now pick all edges one by one from sorted list of edges

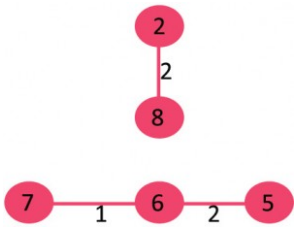
1. *Pick edge 7-6:* No cycle is formed, include it.



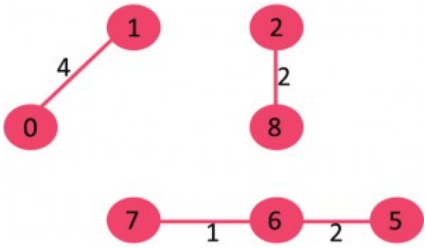
2. *Pick edge 8-2:* No cycle is formed, include it.



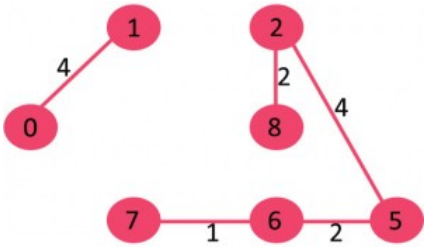
3. *Pick edge 6-5:* No cycle is formed, include it.



4. *Pick edge 0-1:* No cycle is formed, include it.

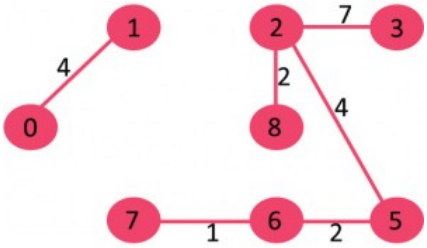


5. *Pick edge 2-5:* No cycle is formed, include it.



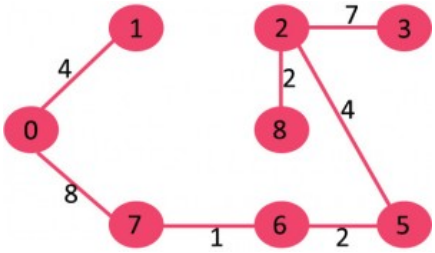
6. *Pick edge 8-6:* Since including this edge results in cycle, discard it.

7. *Pick edge 2-3:* No cycle is formed, include it.



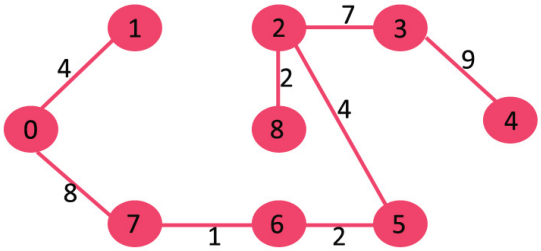
8. *Pick edge 7-8:* Since including this edge results in cycle, discard it.

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.

C/C++

```
 // 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 = new Graph;
    graph->V = V;
    graph->E = E;
```

```
    graph->edge = new Edge[E];

    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];  // Tnis 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 ssubsets
    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;
}
```

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];  // Tnis 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 ssubsets
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 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
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
```

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])

    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 does'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))
```

```
# Driver code
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


Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
```

Time Complexity: O(ElogE) or O(ElogV). Sorting of edges takes O(ELogE) time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost O(LogV) time. So overall complexity is O(ELogE + ELogV) time. The value of E can be atmost $O(V^2)$, so O(LogV) are O(LogE) same. Therefore, overall time complexity is O(ElogE) or O(ElogV)

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.

Source

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Chapter 48

Largest gap in an array

Largest gap in an array - GeeksforGeeks

Given an unsorted array of length N and we have to find largest gap between any two elements of array.In simple words, find max(A_i-A_j) where 1 i N and 1 j N.

Examples:

```
Input : arr = {3, 10, 6, 7}
Output : 7
Explanation :
Here, we can see largest gap can be
found between 3 and 10 which is 7

Input : arr = {-3, -1, 6, 7, 0}
Output : 10
Explanation :
Here, we can see largest gap can be
found between -3 and 7 which is 10
```

Simple Approach:
A simple solution is, we can use naive approach.We will check absolute difference of every pair in the array and we will find the maximum value of it.So we will run two loops one is for i and one is for j complexity of this method is O(N^2)

C

```
 // A C program to find largest gap
// between two elements in an array.
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
```

```
// function to solve the given problem
int solve(int a[], int n)
{
    int max1 = INT_MIN;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (abs(a[i] - a[j]) > max1) {
                max1 = abs(a[i] - a[j]);
            }
        }
    }
    return max1;
}

int main()
{
    int arr[] = { -1, 2, 3, -4, -10, 22 };
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Largest gap is : %d", solve(arr, size));
    return 0;
}
```

Java

```
 // A Java program to find
// largest gap between
// two elements in an array.
import java .io.*;

class GFG
{

// function to solve
// the given problem
static int solve(int []a,
                 int n)
{
    int max1 = Integer.MIN_VALUE ;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Math.abs(a[i] -
                         a[j]) > max1)
            {
                max1 = Math.abs(a[i] -
                                a[j]);
            }
```

```
        }
    }
    return max1;
}

// Driver Code
static public void main (String[] args)
{
    int []arr = {-1, 2, 3,
                 -4, -10, 22};
    int size = arr.length;
    System.out.println("Largest gap is : " +
                        solve(arr, size));
}
}

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

C#

```
 // A C# program to find
// largest gap between
// two elements in an array.
using System;

class GFG
{

// function to solve
// the given problem
static int solve(int []a,
                 int n)
{
    int max1 = int.MinValue ;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Math.Abs(a[i] -
                         a[j]) > max1)
            {
                max1 = Math.Abs(a[i] -
                                a[j]);
            }
        }
    }
    return max1;
```

```
}

// Driver Code
static public void Main ()
{
    int []arr = {-1, 2, 3,
                 -4, -10, 22};
    int size = arr.Length;
    Console.WriteLine("Largest gap is : " +
                          solve(arr, size));
}
}

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

PHP

```
 <?php
// A PHP program to find
// largest gap between
// two elements in an array.

// function to solve
// the given problem
function solve($a, $n)
{
    $max1 = PHP_INT_MIN;
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $n; $j++)
        {
            if (abs($a[$i] -
                    $a[$j]) > $max1)
            {
                $max1 = abs($a[$i] -
                            $a[$j]);
            }
        }
    }
    return $max1;
}

// Driver Code
$arr = array(-1, 2, 3,
             -4, -10, 22);
$size = count($arr);
echo "Largest gap is : ",
```

```
      solve($arr, $size);

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

Output:

```
Largest gap is : 32
```

Better Approach:
Now we will see a better approach it is greedy approach which can solve this problem in O(N). we will find maximum and minimum element of the array which can be done in O(N) and then we will return value of (maximum-minimum).
C

```
 // A C program to find largest gap between
// two elements in an array.
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// function to solve the given problem
int solve(int a[], int n)
{
    int min1 = a[0];
    int max1 = a[0];

    // finding maximum and minimum of an array
    for (int i = 0; i < n; i++) {
        if (a[i] > max1)
            max1 = a[i];
        if (a[i] < min1)
            min1 = a[i];
    }

    return abs(min1 - max1);
}

int main()
{
    int arr[] = { -1, 2, 3, 4, -10 };
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Largest gap is : %d", solve(arr, size));
    return 0;
}
```

Java

```
 // A Java program to find largest gap
// between two elements in an array.
import java.io.*;

class GFG {

    // function to solve the given
    // problem
    static int solve(int a[], int n)
    {
        int min1 = a[0];
        int max1 = a[0];

        // finding maximum and minimum
        // of an array
        for (int i = 0; i < n; i++)
        {
            if (a[i] > max1)
                max1 = a[i];
            if (a[i] < min1)
                min1 = a[i];
        }

        return Math.abs(min1 - max1);
    }

    // Driver code
    public static void main (String[] args)
    {
        int []arr = { -1, 2, 3, 4, -10 };
        int size = arr.length;
        System.out.println("Largest gap is : "
                           + solve(arr, size));
    }
}

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

C#

```
 // A C# program to find
// largest gap between
// two elements in an array.
using System;
```

```
class GFG
{

    // function to solve
    // the given problem
    static int solve(int []a,
                     int n)
    {
        int min1 = a[0];
        int max1 = a[0];

        // finding maximum and
        // minimum of an array
        for (int i = 0; i < n; i++)
        {
            if (a[i] > max1)
                max1 = a[i];
            if (a[i] < min1)
                min1 = a[i];
        }

        return Math.Abs(min1 -
                        max1);
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {-1, 2, 3, 4, -10};
        int size = arr.Length;
        Console.WriteLine("Largest gap is : " +
                              solve(arr, size));
    }
}

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

PHP

```
 <?php
// A PHP program to find
// largest gap between
// two elements in an array.

// function to solve
// the given problem
function solve($a, $n)
```

```
{
    $min1 = $a[0];
    $max1 = $a[0];

    // finding maximum and
    // minimum of an array
    for ($i = 0; $i < $n; $i++)
    {
        if ($a[$i] > $max1)
            $max1 = $a[$i];
        if ($a[$i] < $min1)
            $min1 = $a[$i];
    }

    return abs($min1 - $max1);
}

// Driver Code
$arr = array(-1, 2, 3, 4, -10);
$size = count($arr);
echo "Largest gap is : ",
      solve($arr, $size);

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

Output:

```
Largest gap is : 14
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/largest-gap-in-an-array/

Chapter 49

Largest lexicographic array with at-most K consecutive swaps

Largest lexicographic array with at-most K consecutive swaps - GeeksforGeeks

Given an array arr[], find the lexicographically largest array that can be obtained by performing at-most k consecutive swaps.

Examples :

```
Input : arr[] = {3, 5, 4, 1, 2}
        k = 3
Output : 5, 4, 3, 2, 1
Explanation : Array given : 3 5 4 1 2
After swap 1 : 5 3 4 1 2
After swap 2 : 5 4 3 1 2
After swap 3 : 5 4 3 2 1

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

Brute Force Approach : Generate all permutation of the array and then pick the one which satisfies the condition of at most K swaps. The time complexity of this approach is **O(n!)**.

Optimized Approach : In this greedy approach, first find the largest element present in the array which is greater than(if the 1st position element is not the greatest) the 1st position and which can be placed at the 1st position with at-most K swaps. After finding that element, note its index. Then, swap elements of the array and update K value. Apply this procedure for other positions till k is non-zero or array becomes lexicographically largest.

Below is the implementation of above approach :

C++

```
 // C++ program to find lexicographically
// maximum value after k swaps.
#include <bits/stdc++.h>
using namespace std;

// Function which modifies the array
void KSwapMaximum(int arr[], int n, int k)
{
    for (int i = 0; i < n - 1 && k > 0; ++i) {

        // Here, indexPositionition is set where we
        // want to put the current largest integer
        int indexPosition = i;
        for (int j = i + 1; j < n; ++j) {

            // If we exceed the Max swaps
            // then break the loop
            if (k <= j - i)
                break;

            // Find the maximum value from i+1 to
            // max k or n which will replace
            // arr[indexPosition]
            if (arr[j] > arr[indexPosition])
                indexPosition = j;
        }

        // Swap the elements from Maximum indexPosition
        // we found till now to the ith index
        for (int j = indexPosition; j > i; --j)
            swap(arr[j], arr[j - 1]);

        // Updates k after swapping indexPosition-i
        // elements
        k -= indexPosition - i;
    }
}

// Driver code
int main()
{
    int arr[] = { 3, 5, 4, 1, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;

    KSwapMaximum(arr, n, k);
```

```
    // Print the final Array
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
}
```

Java

```
 // Java program to find
// lexicographically
// maximum value after
// k swaps.
import java.io.*;

class GFG
{
    static void SwapInts(int array[],
                         int position1,
                         int position2)
    {
        // Swaps elements
        // in an array.

        // Copy the first
        // position's element
        int temp = array[position1];

        // Assign to the
        // second element
        array[position1] = array[position2];

        // Assign to the
        // first element
        array[position2] = temp;
    }

    // Function which
    // modifies the array
    static void KSwapMaximum(int []arr,
                             int n, int k)
    {
        for (int i = 0;
                 i < n - 1 && k > 0; ++i)
        {

            // Here, indexPositionition
            // is set where we want to
            // put the current largest
```

```
            // integer
            int indexPosition = i;
            for (int j = i + 1; j < n; ++j)
            {

                // If we exceed the
                // Max swaps then
                // break the loop
                if (k <= j - i)
                    break;

                // Find the maximum value
                // from i+1 to max k or n
                // which will replace
                // arr[indexPosition]
                if (arr[j] > arr[indexPosition])
                    indexPosition = j;
            }

            // Swap the elements from
            // Maximum indexPosition
            // we found till now to
            // the ith index
            for (int j = indexPosition; j > i; --j)
                SwapInts(arr, j, j - 1);

            // Updates k after swapping
            // indexPosition-i elements
            k -= indexPosition - i;
        }
    }

    // Driver code
    public static void main(String args[])
    {
        int []arr = { 3, 5, 4, 1, 2 };
        int n = arr.length;
        int k = 3;

        KSwapMaximum(arr, n, k);

        // Print the final Array
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
    }
}

// This code is contributed by
```

```
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python program to find
# lexicographically
# maximum value after
# k swaps.

arr = [3, 5, 4, 1, 2]

# Function which
# modifies the array
def KSwapMaximum(n, k) :

    global arr
    for i in range(0, n - 1) :
        if (k > 0) :

            # Here, indexPositionition
            # is set where we want to
            # put the current largest
            # integer
            indexPosition = i
            for j in range(i + 1, n) :

                # If we exceed the Max swaps
                # then break the loop
                if (k <= j - i) :
                    break

                # Find the maximum value
                # from i+1 to max k or n
                # which will replace
                # arr[indexPosition]
                if (arr[j] > arr[indexPosition]) :
                    indexPosition = j

            # Swap the elements from
            # Maximum indexPosition
            # we found till now to
            # the ith index
            for j in range(indexPosition, i, -1) :
                t = arr[j]
                arr[j] = arr[j - 1]
                arr[j - 1] = t

            # Updates k after swapping
```

```
            # indexPosition-i elements
            k = k - indexPosition - i

# Driver code
n = len(arr)
k = 3

KSwapMaximum(n, k)

# Print the final Array
for i in range(0, n) :
    print ("{} " .
            format(arr[i]),
                 end = "")

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find
// lexicographically
// maximum value after
// k swaps.
using System;

class GFG
{
    static void SwapInts(int[] array,
                         int position1,
                         int position2)
    {
        // Swaps elements in an array.

        // Copy the first position's element
        int temp = array[position1];

        // Assign to the second element
        array[position1] = array[position2];

        // Assign to the first element
        array[position2] = temp;
    }

    // Function which
    // modifies the array
    static void KSwapMaximum(int []arr,
                             int n, int k)
```

```
{
    for (int i = 0;
             i < n - 1 && k > 0; ++i)
    {

        // Here, indexPositionition
        // is set where we want to
        // put the current largest
        // integer
        int indexPosition = i;
        for (int j = i + 1; j < n; ++j)
        {

            // If we exceed the
            // Max swaps then
            // break the loop
            if (k <= j - i)
                break;

            // Find the maximum value
            // from i+1 to max k or n
            // which will replace
            // arr[indexPosition]
            if (arr[j] > arr[indexPosition])
                indexPosition = j;
        }

        // Swap the elements from
        // Maximum indexPosition
        // we found till now to
        // the ith index
        for (int j = indexPosition; j > i; --j)
            SwapInts(arr, j, j - 1);

        // Updates k after swapping
        // indexPosition-i elements
        k -= indexPosition - i;
    }
}

// Driver code
static void Main()
{
    int []arr = new int[]{ 3, 5, 4, 1, 2 };
    int n = arr.Length;
    int k = 3;

    KSwapMaximum(arr, n, k);
```

```
        // Print the final Array
        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");
    }
}
// This code is contributed by
// Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP program to find
// lexicographically
// maximum value after
// k swaps.

function swap(&$x, &$y)
{
    $x ^= $y ^= $x ^= $y;
}

// Function which
// modifies the array
function KSwapMaximum(&$arr, $n, $k)
{
    for ($i = 0;
         $i < $n - 1 &&
         $k > 0; $i++)
    {

        // Here, indexPositionition
        // is set where we want to
        // put the current largest
        // integer
        $indexPosition = $i;
        for ($j = $i + 1;
             $j < $n; $j++)
        {

            // If we exceed the Max swaps
            // then break the loop
            if ($k <= $j - $i)
                break;

            // Find the maximum value
            // from i+1 to max k or n
            // which will replace
```

```
            // arr[indexPosition]
            if ($arr[$j] > $arr[$indexPosition])
                $indexPosition = $j;
        }

        // Swap the elements from
        // Maximum indexPosition
        // we found till now to
        // the ith index
        for ($j = $indexPosition;
             $j > $i; $j--)
            swap($arr[$j], $arr[$j - 1]);

        // Updates k after swapping
        // indexPosition-i elements
        $k -= $indexPosition - $i;
    }
}

// Driver code
$arr = array( 3, 5, 4, 1, 2 );
$n = count($arr);
$k = 3;

KSwapMaximum($arr, $n, $k);

// Print the final Array
for ($i = 0; $i < $n; $i++)
    echo ($arr[$i]." ");

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
5 4 3 1 2
```

Time Complexity: O(N*N)
Auxilliary Space: O(1)

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/largest-lexicographic-array-with-at-most-k-consecutive-swaps/

Chapter 50

Largest palindromic number by permuting digits

Largest palindromic number by permuting digits - GeeksforGeeks

Given N(very large), task if to print the largest palindromic number obtained by permuting the digits of N. If it is not possible to make a palindromic number, then print an appropriate message.

Examples :

```
Input : 313551
Output : 531135
Explanations : 531135 is the largest number
which is a palindrome, 135531, 315513 and other
numbers can also be formed but we need the highest
of all of the palindromes.

Input : 331
Output : 313

Input : 3444
Output : Pallindrome cannot be formed
```

Naive Approach : The **naive** approach will be to try all the permutations possible, and print the largest of such combinations which is a palindrome.

Efficient Approach : An efficient approach will be to use **Greedy algorithm.** Since the number is large, store the number in a string. Store the count of occurrences of every digit in the given number in a map. Check if it is possible to form a palindrome or not. If the digits of the given number can be rearranged to form a palindrome, then apply the greedy approach to obtain the number. Check for the occurrence of every digit (9 to 0), and place

every available digit at front and back. **Initially, the front pointer will be at index 0, as the largest digit will be placed at first to make the number a large one.** With every step, move the front pointer 1 position ahead. If the digit occurs an odd number of times, then place **one digit in the middle and rest of the even number of digits at front and back**. Keep repeating the process **(map[digit]/2)** number of times for a single digit. After placing a particular digit which occurs an even number of times at the front and back, move the front pointer one step ahead. The placing is done till map[digit] is 0. The char array will have the largest palindromic number possible after completion of the placing of digits greedily.

In the worst case, the time complexity will be **O(10 * (length of string/2))**, in case the number consists of a same digit at every position.

Below is the implementation of the above idea :

```
 // CPP program to print the largest palindromic
// number by permuting digits of a number
#include <bits/stdc++.h>
using namespace std;

// function to check if a number can be
// permuted to form a palindrome number
bool possibility(unordered_map<int, int> m,
                 int length, string s)
{
    // counts the occurrence of number which is odd
    int countodd = 0;
    for (int i = 0; i < length; i++) {

        // if occurrence is odd
        if (m[s[i] - '0'] & 1)
            countodd++;

        // if number exceeds 1
        if (countodd > 1)
            return false;
    }

    return true;
}

// function to print the largest palindromic number
// by permuting digits of a number
void largestPalindrome(string s)
{

    // string length
    int l = s.length();
```

```
// map that marks the occurrence of a number
unordered_map<int, int> m;
for (int i = 0; i < l; i++)
    m[s[i] - '0']++;

// check the possibility of a palindromic number
if (possibility(m, l, s) == false) {
    cout << "Palindrome cannot be formed";
    return;
}

// string array that stores the largest
// permuted palindromic number
char largest[l];

// pointer of front
int front = 0;

// greedily start from 9 to 0 and place the
// greater number in front and odd in the
// middle
for (int i = 9; i >= 0; i--) {

    // if the occurrence of number is odd
    if (m[i] & 1) {

        // place one odd occurring number
        // in the middle
        largest[l / 2] = char(i + 48);

        // decrease the count
        m[i]--;

        // place the rest of numbers greedily
        while (m[i] > 0) {
            largest[front] = char(i + 48);
            largest[l - front - 1] = char(i + 48);
            m[i] -= 2;
            front++;
        }
    }
    else {

        // if all numbers occur even times,
        // then place greedily
        while (m[i] > 0) {

            // place greedily at front
```

```
                largest[front] = char(i + 48);
                largest[l - front - 1] = char(i + 48);

                // 2 numbers are placed, so decrease the count
                m[i] -= 2;

                // increase placing position
                front++;
            }
        }
    }

    // print the largest string thus formed
    for (int i = 0; i < l; i++)
        cout << largest[i];
}

// Driver Code
int main()
{
    string s = "313551";
    largestPalindrome(s);
    return 0;
}
```

Output:

```
531135
```

Time Complexity : O(n), where n is the length of string.

Source

https://www.geeksforgeeks.org/largest-palindromic-number-permuting-digits/

Chapter 51

Largest permutation after at most k swaps

Largest permutation after at most k swaps - GeeksforGeeks

Given a permutation of first **n** natural numbers as array and an integer k. Print the lexicographically largest permutation after at most k swaps

```
Input: arr[] = {4, 5, 2, 1, 3}
       k = 3
Output: 5 4 3 2 1
Explanation:
Swap 1st and 2nd elements: 5 4 2 1 3
Swap 3rd and 5th elements: 5 4 3 1 2
Swap 4th and 5th elements: 5 4 3 2 1

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

A **Naive approach** is to one by one generate permutation in lexicographically decreasing order. Compare every generated permutation with original array and count the number of swaps required to convert. If count is less than or equal to k, print this permutation. Problem of this approach is that it would be difficult to implement and will definitely time out for large value of N.

An **Efficient** approach is to think greedily. If we visualize the problem then we will get to know that largest permutation can only be obtained if it starts with n and continues with n-1, n-2,…. So we just need to put the 1^{st}, 2^{nd}, 3^{rd}, …, k^{th} largest element to their respective position.
C++

```
 // Below is C++ code to print largest permutation
// after atmost K swaps
#include<bits/stdc++.h>
using namespace std;

// Function to calculate largest permutation after
// atmost K swaps
void KswapPermutation(int arr[], int n, int k)
{
    // Auxiliary dictionary of storing the position
    // of elements
    int pos[n+1];

    for (int i = 0; i < n; ++i)
        pos[arr[i]] = i;

    for (int i=0; i<n && k; ++i)
    {
        // If element is already i'th largest,
        // then no need to swap
        if (arr[i] == n-i)
            continue;

        // Find position of i'th largest value, n-i
        int temp = pos[n-i];

        // Swap the elements position
        pos[arr[i]] = pos[n-i];
        pos[n-i] = i;

        // Swap the ith largest value with the
        // current value at ith place
        swap(arr[temp], arr[i]);

        // decrement number of swaps
        --k;
    }
}

// Driver code
int main()
{
    int arr[] = {4, 5, 2, 1, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;

    KswapPermutation(arr, n, k);
    cout << "Largest permutation after "
```

```
         << k << " swaps:n";
    for (int i=0; i<n; ++i)
        printf("%d ", arr[i]);
    return 0;
}
```

Java

```
 // Below is Java code to print largest permutation
// after atmost K swaps
class GFG {

    // Function to calculate largest permutation after
    // atmost K swaps
    static void KswapPermutation(int arr[], int n, int k)
    {

        // Auxiliary dictionary of storing the position
        // of elements
        int pos[] = new int[n+1];

        for (int i = 0; i < n; ++i)
            pos[arr[i]] = i;

        for (int i = 0; i < n && k > 0; ++i)
        {

            // If element is already i'th largest,
            // then no need to swap
            if (arr[i] == n-i)
                continue;

            // Find position of i'th largest value, n-i
            int temp = pos[n-i];

            // Swap the elements position
            pos[arr[i]] = pos[n-i];
            pos[n-i] = i;

            // Swap the ith largest value with the
            // current value at ith place
            int tmp1 = arr[temp];
            arr[temp] = arr[i];
            arr[i] = tmp1;

            // decrement number of swaps
            --k;
        }
```

```
    }

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

        int arr[] = {4, 5, 2, 1, 3};
        int n = arr.length;
        int k = 3;

        KswapPermutation(arr, n, k);

        System.out.print("Largest permutation "
                  + "after " + k + " swaps:\n");

        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
    }
}

// This code is contributed by Anant Agarwal.
```

Python

```
 # Python code to print largest permutation after K swaps

def KswapPermutation(arr, n, k):

    # Auxiliary array of storing the position of elements
    pos = {}
    for i in range(n):
        pos[arr[i]] = i

    for i in range(n):

        # If K is exhausted then break the loop
        if k == 0:
            break

        # If element is already largest then no need to swap
        if (arr[i] == n-i):
            continue

        # Find position of i'th largest value, n-i
        temp = pos[n-i]

        # Swap the elements position
        pos[arr[i]] = pos[n-i]
```

```
        pos[n-i] = i

        # Swap the ith largest value with the value at
        # ith place
        arr[temp], arr[i] = arr[i], arr[temp]

        # Decrement K after swap
        k = k-1

# Driver code
arr = [4, 5, 2, 1, 3]
n = len(arr)
k = 3
KswapPermutation(arr, N, K)

# Print the answer
print "Largest permutation after", K, "swaps: "
print " ".join(map(str,arr))
```

C#

```
 // Below is C# code to print largest
// permutation after atmost K swaps.
using System;

class GFG {

    // Function to calculate largest
    // permutation after atmost K
    // swaps
    static void KswapPermutation(int []arr,
                                 int n, int k)
    {

        // Auxiliary dictionary of storing
        // the position of elements
        int []pos = new int[n+1];

        for (int i = 0; i < n; ++i)
            pos[arr[i]] = i;

        for (int i = 0; i < n && k > 0; ++i)
        {

            // If element is already i'th
            // largest, then no need to swap
            if (arr[i] == n-i)
                continue;
```

```
            // Find position of i'th largest
            // value, n-i
            int temp = pos[n-i];

            // Swap the elements position
            pos[arr[i]] = pos[n-i];
            pos[n-i] = i;

            // Swap the ith largest value with
            // the current value at ith place
            int tmp1 = arr[temp];
            arr[temp] = arr[i];
            arr[i] = tmp1;

            // decrement number of swaps
            --k;
        }
    }

    // Driver method
    public static void Main()
    {

        int []arr = {4, 5, 2, 1, 3};
        int n = arr.Length;
        int k = 3;

        KswapPermutation(arr, n, k);

        Console.Write("Largest permutation "
                + "after " + k + " swaps:\n");

        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");
    }
}

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

```
Output:
Largest permutation after 3 swaps:
5 4 3 2 1
```

Time complexity: O(n)
Auxiliary space: O(n)

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/largest-permutation-k-swaps/

Chapter 52

Length and Breadth of rectangle such that ratio of Area to diagonal^2 is maximum

Length and Breadth of rectangle such that ratio of Area to diagonal^2 is maximum - GeeksforGeeks

Given an array of positive integers. The task is to choose a pair of elements from the given array such that they represent the length and breadth of a rectangle and the ratio of its area and its diagonal^2 is maximum.

Note: The array must contains all sides of the rectangle. That is you can choose elements from array which appears atleast twice as a rectangle has two sides of same length and two sides of same breadth.

Examples:

> **Input:** arr[] = {4, 3, 5, 4, 3, 5, 7}
> **Output:** 5, 4
> Among all pairs of length and breadth 5, 4 will generate maximum ratio of Area to Diameter^2.
>
> **Input:** arr[] = {2, 2, 2, 2, 2, 2}
> **Output:** 2, 2
> There is only one possible pair of length and breadth 2, 2 which will generate maximum ratio of Area to Diameter^2.

Given below are some properties of the rectangle that are to be satisfied in order to form it.

- A rectangle can only be formed when we have at least pair of equal integers. An integer occurring once can't be a part of any rectangle. So, in our solution, we will consider only the integers whose occurrence is more than once.

- The ratio of Area and its diameter [2] is $\mathbf{lb/(l^2 + b^2)}$ is a monotonic decreasing function in term of one variable. This means if we are fixing the length then as we will decrease the breadth the ratio got decreases and accordingly same for fixed breadth. Taking advantage of this we need not iterate the whole array for finding the length for a fixed breadth.

Algorithm :

1. Sort the given array.
2. Create an array (arr_pairs[]) of integers which occur twice in array.
3. Set length = arr_pairs[0], breadth = arr_pairs[0].
4. Iterate from i=2 to sizeof (arr_pairs)
   - if (length/breadth + breadth/length > arr_pairs[i]/arr_pairs[i-1] + arr_pairs[i-1]/arr_pairs[i])
     - update length = arr_pairs[i], breadth = arr_pairs[i-1]
5. Print length and breadth.

Below is the implementation of the above approach.

```
 // CPP for finding maximum
// p^2/A ratio of rectangle
#include <bits/stdc++.h>
using namespace std;

// function to print length and breadth
void findLandB(int arr[], int n)
{
    // sort the input array
    sort(arr, arr + n);

    // create array vector of integers occuring in pairs
    vector<double> arr_pairs;
    for (int i = 0; i < n; i++) {

        // push the same pairs
        if (arr[i] == arr[i + 1]) {
            arr_pairs.push_back(arr[i]);
            i++;
        }
    }

    double length = arr_pairs[0];
    double breadth = arr_pairs[1];
```

```
    double size = arr_pairs.size();

    // calculate length and breadth as per requirement
    for (int i = 2; i < size; i++) {

        // check for given condition
        if ((length / breadth + breadth / length) > (arr_pairs[i] / arr_pairs[i - 1] + arr_pairs

            length = arr_pairs[i];
            breadth = arr_pairs[i - 1];
        }
    }

    // print the required answer
    cout << length << ", " << breadth << endl;
}

// Driver Code
int main()
{
    int arr[] = { 4, 2, 2, 2, 5, 6, 5, 6, 7, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findLandB(arr, n);
    return 0;
}
```

Output:

```
2, 2
```

Time Complexity: O(N * log N)

Source

https://www.geeksforgeeks.org/length-and-breadth-of-rectangle-such-that-ratio-of-area-to-diagonal2-is-maximum/

Chapter 53

Lexicographically largest subsequence such that every character occurs at least k times

Lexicographically largest subsequence such that every character occurs at least k times - GeeksforGeeks

Given a string **S** and an integer **K**. The task is to find lexicographically largest subsequence of S, say T, such that every character in T must occur at least K times.

Examples:

```
Input : S = "banana", K = 2.
Output : nn
Possible subsequence where each character exists at least 2 times are:

From the above subsequences, "nn" is the lexicographically largest.
```

The idea is to solve greedily the above problem. If we want to make the subsequence lexicographically largest, we must give priority to lexicographically larger characters. 'z' is the largest character, let suppose z occurs f_z times in S. If $f_z >= K$, append 'z'z k times in the string T and keep removing characters from the left of S until all the z's are removed. Apply the strategy with 'y', 'w', ..., 'a'. In the end you will find the answer.

Let see an example. Suppose S = "zzwzawa" and K = 2. Start with the largest character 'z'. Here $f_z = 3 >= K$. So T will become "zzz" and we will remove letters from the left of S until all the z's are removed. So now S will became "awa". Next largest is 'y' but that occurs 0 times in k so we will skip it. We will skip 'w', 'v' etc also until we go to 'a' which occurs 2 times. Now T will become "zzzaa" and S will become a empty string. Our answer is "zzzaa".

Below is C++ implementation of this approach:

```
 // C++ program to find lexicographically largest
// subsequence where every character appears at
// least k times.
#include <bits/stdc++.h>
using namespace std;

// Find lexicographically largest subsequence of
// s[0..n-1] such that every character appears
// at least k times. The result is filled in t[]
void subsequence(char s[], char t[], int n, int k)
{
    int last = 0, cnt = 0, new_last = 0, size = 0;

    // Starting from largest charter 'z' to 'a'
    for (char ch = 'z'; ch >= 'a'; ch--) {
        cnt = 0;

        // Counting the frequency of the character
        for (int i = last; i < n; i++) {
            if (s[i] == ch)
                cnt++;
        }

        // If frequency is greater than k
        if (cnt >= k) {

            // From the last point we leave
            for (int i = last; i < n; i++) {

                // check if string contain ch
                if (s[i] == ch) {

                    // If yes, append to output string
                    t[size++] = ch;
                    new_last = i;
                }
            }

            // Update the last point.
            last = new_last;
        }
    }
    t[size] = '\0';
}

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

```
    char s[] = "banana";
    int n = sizeof(s);
    int k = 2;
    char t[n];
    subsequence(s, t, n - 1, k);
    cout << t << endl;
    return 0;
}
```

Output:

```
nn
```

Source

https://www.geeksforgeeks.org/lexicographically-largest-subsequence-every-character-occurs-least-k-times/

Chapter 54

Lexicographically smallest array after at-most K consecutive swaps

Lexicographically smallest array after at-most K consecutive swaps - GeeksforGeeks

Given an array arr[], find the lexicographically smallest array that can be obtained after performing at maximum of k consecutive swaps.
Examples :

```
Input: arr[] = {7, 6, 9, 2, 1}
        k = 3
Output: arr[] = {2, 7, 6, 9, 1}
Explanation: Array is: 7, 6, 9, 2, 1
Swap 1:   7, 6, 2, 9, 1
Swap 2:   7, 2, 6, 9, 1
Swap 3:   2, 7, 6, 9, 1
So Our final array after k = 3 swaps :
2, 7, 6, 9, 1

Input: arr[] = {7, 6, 9, 2, 1}
        k = 1
Output: arr[] = {6, 7, 9, 2, 1}
```

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

Naive approach is to generate all the permutation of array and pick the smallest one which satisfy the condition of at-most k swaps. Time complexity of this approach is $\Omega(n!)$, which will definitely time out for large value of n.

An **Efficient** approach is to think greedily. We first pick the smallest element from array a_1, a_2, a_3…(a_k or a_n) [We consider a_k when k is smaller, else n]. We place the smallest element

to the a_0 position after shifting all these elements by 1 position right. We subtract number of swaps (number of swaps is number of shifts minus 1) from k. If still we are left with k > 0 then we apply the same procedure from the very next starting position i.e., a_2, a_3,…(a_k or a_n) and then place it to the a_1 position. So we keep applying the same process until k becomes 0.

C++

```
 // C++ program to find lexicographically minimum
// value after k swaps.
#include<bits/stdc++.h>
using namespace std ;

// Modifies arr[0..n-1] to lexicographically smallest
// with k swaps.
void minimizeWithKSwaps(int arr[], int n, int k)
{
    for (int i = 0; i<n-1 && k>0; ++i)
    {
        // Set the position where we want
        // to put the smallest integer
        int pos = i;
        for (int j = i+1; j<n ; ++j)
        {
            // If we exceed the Max swaps
            // then terminate the loop
            if (j-i > k)
                break;

            // Find the minimum value from i+1 to
            // max k or n
            if (arr[j] < arr[pos])
                pos = j;
        }

        // Swap the elements from Minimum position
        // we found till now to the i index
        for (int j = pos; j>i; --j)
            swap(arr[j], arr[j-1]);

        // Set the final value after swapping pos-i
        // elements
        k -=  pos-i;
    }
}

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

```
    int arr[] = {7, 6, 9, 2, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;

    minimizeWithKSwaps(arr, n, k);

    //Print the final Array
    for (int i=0; i<n; ++i)
        cout << arr[i] <<" ";
}
```

Java

```
 // Java program to find lexicographically minimum
// value after k swaps.
class GFG {

    // Modifies arr[0..n-1] to lexicographically
    // smallest with k swaps.
    static void minimizeWithKSwaps(int arr[], int n, int k)
    {
        for (int i = 0; i < n-1 && k > 0; ++i)
        {

            // Set the position where we want
            // to put the smallest integer
            int pos = i;
            for (int j = i+1; j < n ; ++j)
            {

                // If we exceed the Max swaps
                // then terminate the loop
                if (j - i > k)
                    break;

                // Find the minimum value from i+1 to
                // max k or n
                if (arr[j] < arr[pos])
                    pos = j;
            }

            // Swap the elements from Minimum position
            // we found till now to the i index
            int temp;

            for (int j = pos; j>i; --j)
            {
                temp=arr[j];
```

```
                arr[j]=arr[j-1];
                arr[j-1]=temp;
            }

            // Set the final value after swapping pos-i
            // elements
            k -= pos-i;
        }
    }

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

        int arr[] = {7, 6, 9, 2, 1};
        int n = arr.length;
        int k = 3;

        minimizeWithKSwaps(arr, n, k);

        //Print the final Array
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] +" ");
    }
}

// This code is contributed by Anant Agarwal.
```

Python

```
 # Python program to find lexicographically minimum
# value after k swaps.
def minimizeWithKSwaps(arr, n, k):

    for i in range(n-1):

        # Set the position where we we want
    # to put the smallest integer
        pos = i
        for j in range(i+1, n):

            # If we exceed the Max swaps
        # then terminate the loop
            if (j-i > k):
                break

            # Find the minimum value from i+1 to
            # max (k or n)
```

```
            if (arr[j] < arr[pos]):
                pos = j

        # Swap the elements from Minimum position
        # we found till now to the i index
        for j in range(pos, i, -1):
            arr[j],arr[j-1] = arr[j-1], arr[j]

        # Set the final value after swapping pos-i
        # elements
        k -= pos - i


# Driver Code
n, k = 5, 3
arr = [7, 6, 9, 2, 1]
minimizeWithKSwaps(arr, n, k)

# Print the final Array
for i in range(n):
    print(arr[i], end = " ")
```

C#

```
 // C# program to find lexicographically
// minimum value after k swaps.
using System;

class GFG {

    // Modifies arr[0..n-1] to lexicographically
    // smallest with k swaps.
    static void minimizeWithKSwaps(int []arr, int n,
                                            int k)
    {
        for (int i = 0; i < n-1 && k > 0; ++i)
        {
            // Set the position where we want
            // to put the smallest integer
            int pos = i;
            for (int j = i+1; j < n ; ++j)
            {
                // If we exceed the Max swaps
                // then terminate the loop
                if (j - i > k)
                    break;

                // Find the minimum value from
```

```
                // i + 1 to max k or n
                if (arr[j] < arr[pos])
                    pos = j;
            }

            // Swap the elements from Minimum position
            // we found till now to the i index
            int temp;

            for (int j = pos; j>i; --j)
            {
                temp=arr[j];
                arr[j]=arr[j-1];
                arr[j-1]=temp;
            }

            // Set the final value after
            // swapping pos-i elements
            k -= pos-i;
        }
    }

    // Driver method
    public static void Main()
    {
        int []arr = {7, 6, 9, 2, 1};
        int n = arr.Length;
        int k = 3;

        // Function calling
        minimizeWithKSwaps(arr, n, k);

        // Print the final Array
        for (int i=0; i<n; ++i)
          Console.Write(arr[i] +" ");
    }
}

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

php

```
 <?php
// php program to find lexicographically minimum
// value after k swaps.

// Modifies arr[0..n-1] to lexicographically
// smallest with k swaps.
```

```
function minimizeWithKSwaps($arr, $n, $k)
{
    for ($i = 0; $i < $n-1 && $k > 0; ++$i)
    {
        // Set the position where we want
        // to put the smallest integer
        $pos = $i;
        for ($j = $i+1; $j < $n ; ++$j)
        {
            // If we exceed the Max swaps
            // then terminate the loop
            if ($j-$i > $k)
                break;

            // Find the minimum value from
            // i+1 to max k or n
            if ($arr[$j] < $arr[$pos])
                $pos = $j;
        }

        // Swap the elements from Minimum
        // position we found till now to
        // the i index
        for ($j = $pos; $j > $i; --$j)
        {
            $temp = $arr[$j];
            $arr[$j] = $arr[$j-1];
            $arr[$j-1] = $temp;
        }

        // Set the final value after
        // swapping pos-i elements
        $k -= $pos-$i;
    }

    //Print the final Array
    for ($i = 0; $i < $n; ++$i)
        echo $arr[$i] . " ";
}

// Driver code
$arr = array(7, 6, 9, 2, 1);
$n = count($arr);
$k = 3;

minimizeWithKSwaps($arr, $n, $k);

// This code is contributed by Sam007
```

```
?>


Output: 2 7 6 9 1
```

Time complexity: O(N^2)
Auxiliary space: O(1)

Reference:
http://stackoverflow.com/questions/25539423/finding-minimal-lexicographical-array

Improved By : nitin mittal, Sam007

Source

https://www.geeksforgeeks.org/lexicographically-smallest-array-k-consecutive-swaps/

Chapter 55

Lexicographically smallest permutation of a string with given subsequences

Lexicographically smallest permutation of a string with given subsequences - GeeksforGeeks

Given a string consisting only of two lowercase characters x and y and two numbers p and q. The task is to print the lexicographically smallest permutation of the given string such that the count of subsequences of xy is p and of yx is q. If no such string exists, print "Impossible" (without quotes).

Examples:

```
Input: str = "yxxyx", p = 3, q = 3
Output: xyxyx

Input: str = "yxxy", p = 3, q = 2
Output: Impossible
```

Approach: First of all, by induction it can prove that the product of count of 'x' and count of 'y' should be equal to the sum of the count of a subsequence of 'xy' and 'yx' for any given string. If this does not hold then the answer is 'Impossible' else answer always exist.

Now, sort the given string so the count of a subsequence of 'yx' becomes zero. Let **nx** be the count of 'x' and **ny** be count of 'y'. let **a** and **b** be the count of subsequence 'xy' and 'yx' respectively, then **a = nx*ny** and **b = 0**. Then, from beginning of the string find the 'x' which has next 'y' to it and swap both untill you reach end of the string. In each swap **a** is decremented by **1** and **b** is incremented by **1**. Repeat this until the count of a subsequence of 'yx' is achieved i:e **a** becomes **p** and **b** becomes **q**.

Below is the implementation of the above approach:

Source

https://www.geeksforgeeks.org/lexicographically-smallest-permutation-of-a-string-with-given-subsequences/

C++

```
 // CPP program to find lexicographically smallest
// string such that count of subsequence 'xy' and
// 'yx' is p and q respectively.
#include <bits/stdc++.h>
using namespace std;

// function to check if answer exits
int nx = 0, ny = 0;

bool check(string s, int p, int q)
{
    // count total 'x' and 'y' in string
    for (int i = 0; i < s.length(); ++i) {
        if (s[i] == 'x')
            nx++;
        else
            ny++;
    }

    // condition to check existence of answer
    if (nx * ny != p + q)
        return 1;
    else
        return 0;
}

// function to find lexicographically smallest string
string smallestPermutation(string s, int p, int q)
{
    // check if answer exist or not
    if (check(s, p, q) == 1) {
        return "Impossible";
    }

    sort(s.begin(), s.end());
    int a = nx * ny, b = 0, i, j;

    // check if count of 'xy' and 'yx' becomes
    // equal to p and q respectively.
```

```
    if (a == p && b == q) {
        return s;
    }

    // Repeat until answer is found.
    while (1) {
        // Find index of 'x' to swap with 'y'.
        for (i = 0; i < s.length() - 1; ++i) {
            if (s[i] == 'x' && s[i + 1] == 'y')
                break;
        }

        for (j = i; j < s.length() - 1; j++) {
            if (s[j] == 'x' && s[j + 1] == 'y') {
                swap(s[j], s[j + 1]);
                a--; // 'xy' decrement by 1
                b++; // 'yx' increment by 1

                // check if count of 'xy' and 'yx' becomes
                // equal to p and q respectively.
                if (a == p && b == q) {
                    return s;
                }
            }
        }
    }
}

// Driver code
int main()
{
    string s = "yxxyx";
    int p = 3, q = 3;

    cout<< smallestPermutation(s, p, q);

    return 0;
}
```

Java

```
// Java program to find lexicographically
// smallest string such that count of
// subsequence 'xy' and 'yx' is p and
// q respectively.
import java.util.*;

class GFG
{
```

static int nx = 0, ny = 0;

static boolean check(String s,
int p, int q)
{
// count total ‘x’ and ‘y’ in string
for (int i = 0; i < s.length(); ++i) { if (s.charAt(i) == ’x’) nx++; else ny++; } // condition to check // existence of answer if ((nx * ny) != (p + q)) return true; else return false; } public static String smallestPermutation(String s, int p, int q) { if (check(s, p, q) == true) { return ”Impossible”; } char tempArray[] = s.toCharArray(); Arrays.sort(tempArray); String str = new String(tempArray); int a = nx * ny, b = 0, i = 0, j = 0; if (a == p && b == q) { return str; } while (1 > 0)
{

// Find index of ‘x’ to swap with ‘y’.
for (i = 0; i < str.length() - 1; ++i) { if (str.charAt(i) == ’x’ && str.charAt(i + 1) == ’y’) break; } for (j = i; j < str.length() - 1; j++) { if (str.charAt(j) == ’x’ && str.charAt(j + 1) == ’y’) { StringBuilder sb = new StringBuilder(str); sb.setCharAt(j, str.charAt(j + 1)); sb.setCharAt(j + 1, str.charAt(j)); str = sb.toString(); /* char ch[] = str.toCharArray(); char temp = ch[j+1]; ch[j+1] = ch[j]; ch[j] = temp;*/ a--; // ’xy’ decrement by 1 b++; // ’yx’ increment by 1 // check if count of ’xy’ and // ’yx’ becomes equal to p // and q respectively. if (a == p && b == q) { return str; } } } } } // Driver Code public static void main (String[] args) { String s = ”yxxyx”; int p = 3, q = 3; System.out.print(smallestPermutation(s, p, q)); } } // This code is contributed by Kirti_Mangal [tabbyending]

Output:

```
xyxyx
```

Time Complexity: O(N^2)

Improved By : Kirti_Mangal

Chapter 56

Longest subsequence whose average is less than K

Longest subsequence whose average is less than K - GeeksforGeeks

Given an array of N positive integers and Q queries consisting of an integer K, the task is to print the length of the longest subsequence whose average is less than K.

Examples:

> **Input:** a[] = {1, 3, 2, 5, 4}
> Query1: K = 3
> Query2: K = 5
>
> **Output:**
> 4
> 5
> Query1: The subsequence is: {1, 3, 2, 4} or {1, 3, 2, 5}
> Query2: The subsequence is: {1, 3, 2, 5, 4}

A **Naive Approach** is to generate all subsequences using power-set and check for the longest subsequence whose average is less than K.

Time Complexity: $O(2^N * N)$

An **efficient approach** is to sort the array elements and find the average of elements starting from the left. Insert the average of elements computed from the left into the container(vector or arrays). Sort the container's element and then use binary search to search for the number K in the container. The length of the longest subsequence will thus be the index number which upper_bound() returns for every query.

Below is the implementation of the above approach.

```
 // C++ program to perform Q queries
// to find longest subsequence whose
```

```
// average is less than K
#include <bits/stdc++.h>
using namespace std;

// Function to print the length for evey query
int longestSubsequence(int a[], int n, int q[], int m)
{

    // sort array of N elements
    sort(a, a + n);
    int sum = 0;

    // Array to store average from left
    int b[n];

    for (int i = 0; i < n; i++) {
        sum += a[i];
        double av = (double)(sum) / (double)(i + 1);
        b[i] = ((int)(av + 1));
    }

    // Sort aray of average
    sort(b, b + n);

    // number of queries

    for (int i = 0; i < m; i++) {
        int k = q[i];

        // print answer to every query
        // using binary search
        int longest = upper_bound(b, b + n, k) - b;

        cout << "Answer to Query" << i + 1 << ": "
             << longest << endl;
    }
}

// Driver Code
int main()
{
    int a[] = { 1, 3, 2, 5, 4 };
    int n = sizeof(a) / sizeof(a[0]);

    // 4 queries
    int q[] = { 4, 2, 1, 5 };
    int m = sizeof(q) / sizeof(q[0]);
```

```
    longestSubsequence(a, n, q, m);
    return 0;
}
```

Output:

```
Answer to Query1: 5
Answer to Query2: 2
Answer to Query3: 0
Answer to Query4: 5
```

Time Complexity: O(N*log N + M*log N)
Auxiliary Space: O(N)

Source

https://www.geeksforgeeks.org/longest-subsequence-whose-average-is-less-than-k/

Chapter 57

Make array elements equal in Minimum Steps

Make array elements equal in Minimum Steps - GeeksforGeeks

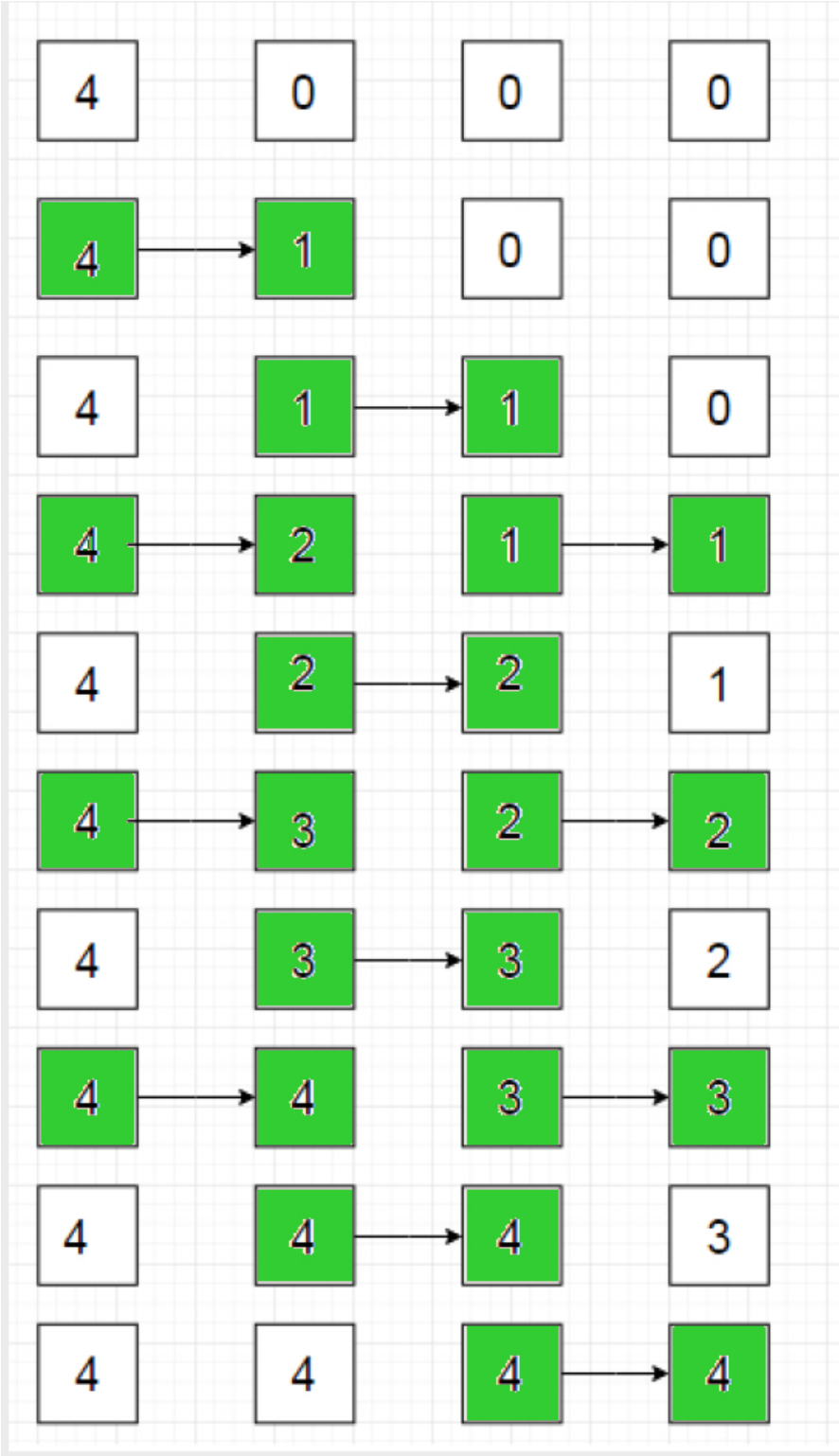
Given an array of **N** elements where the first element is a non zero positive number **M**, and the rest N – 1 elements are 0, the task is to calculate the minimum number of steps required to make the entire array equal while abiding by the following rules:

1. The $\mathbf{i^{th}}$ element can be increased by one if and only if $\mathbf{i\text{-}1^{th}}$ element is strictly greater than the ith element
2. If the $\mathbf{i^{th}}$ element is being increased by one then the $\mathbf{i{+}1^{th}}$ cannot be increased at the same time.(i.e consecutive elements cannot be increased at the same time)
3. Multiple elements can be incremented simultaneously in a single step.

Examples:

Input : N = 3, M = 4
Output : 8
Explanation:
array is 4 0 0
In **4** steps element at index 1 is increased, so the array becomes **{4, 4, 0}**. In the next 4 steps the element at index 3 is increased so array becomes **{4, 4, 4}**
Thus, 4 + 4 = **8** operations are required to make all the array elements equal

Input : N = 4, M = 4
Output : 9
Explanation:
The steps are shown in the flowchart given below
Refer to the flowchart given below.


4 0 0 0
4 1 0 0
4 1 1 0
4 2 1 1
4 2 2 1
4 3 2 2
4 3 3 2
4 4 3 3
4 4 4 3
4 4 4 4

Approach:
To maximise the Number of Increments per Step, more number of Unbalances are created (array[i]>array[i+1]),

Step **1**, element 0 >element 1 so element 1 is incremented,

Step **2**, element 1> element 2 so element 2 is incremented by 1

Step **3**, element 0 > element 1 and element 2> element 3 so element 1 &3 are incremented by 1

Step **4**, element 1 > element 2 element 3 > element 4 so element 2 & 4 are incremented

Step **5**, element 0> element 1; element 2>element 3 ;element 4> element 5; so element 1, 3, &5 are incremented.
and so on…

Consider the following array,
5 0 0 0 0 0

1) **5 1** 0 0 0 0
2) 5 **1 1** 0 0 0
3) **5 2 1 1** 0 0
4) 5 **2 2 1 1** 0
5) **5 3 2 2 1 1**
6) 5 **3 3 2 2** 1
7) **5 4 3 3 2 2**
8) 5 **4 4 3 3** 2
9) **5 5 4 4 3 3**
10) 5 **5 5 4 4** 3
11) 5 5 **5 5 4 4**
12) 5 5 5 **5 5** 4
13) 5 5 5 5 **5 5**

Notice that after an unbalance is created (i.e array[i]>array[i+1]) the element gets incremented by one in alternate steps. In step 1 element 1 gets incremented to 1, in step 2 element 2 gets incremented to 1, in step 3 element 3 gets incremented to 1, so in step n-1, n-1th element will become 1. After that n-1th element is increased by 1 on alternate steps until it reaches the value at element 0. Then the entire array becomes equal.

So the pattern followed by the last element is
(0, 0, 0.., 0) till (N – 4)th element becomes 1 which is **n-4 steps**
and after that,
(0, 0, 1, 1, 2, 2, 3, 3, 4, 4, ... M – 1, M – 1, M) which is **2*m + 1 steps**.

So the Final Result becomes **(N – 3) + 2 * M**

There are a few corner cases which need to be handled, viz. When **N = 1**, array has only a single element, so the number of steps required = 0. and When **N = 2**, number of steps required equal to M
C++

```
// C++ program to make the array elements equal in minimum steps
```

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

// Returns the minumum steps required to make an array of N
// elements equal, where the first array element equals M
int steps(int N, int M)
{
    // Corner Case 1: When N = 1
    if (N == 1)
        return 0;

    // Corner Case 2: When N = 2
    else if (N == 2) // corner case 2
        return M;

    return 2 * M + (N - 3);
}

// Driver Code
int main()
{
    int N = 4, M = 4;
    cout << steps(N, M);
    return 0;
}
```

Java

```
 // Java program to make the array elements
// equal in minimum steps

import java.io.*;

class GFG {

    // Returns the minumum steps required
    // to make an array of N elements equal,
    // where the first array element equals M
    static int steps(int N, int M)
    {
        // Corner Case 1: When N = 1
        if (N == 1)
            return 0;

        // Corner Case 2: When N = 2
        else if (N == 2) // corner case 2
            return M;
```

```
        return 2 * M + (N - 3);
    }

    // Driver Code
    public static void main (String[] args)
    {
        int N = 4, M = 4;
        System.out.print( steps(N, M));
    }
}

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

Python3

```
 # Python program to make
# the array elements equal
# in minimum steps

# Returns the minumum steps
# required to make an array
# of N elements equal, where
# the first array element
# equals M
def steps(N, M):

    # Corner Case 1: When N = 1
    if (N == 1):
        return 0

    # Corner Case 2: When N = 2
    elif(N == 2):
        return M

    return 2 * M + (N - 3)

# Driver Code
N = 4
M = 4
print(steps(N,M))

# This code is contributed
# by Shivi_Aggarwal.
```

C#

```
 // C# program to make the array
```

```
// elements equal in minimum steps
using System;

class GFG
{

    // Returns the minumum steps
    // required to make an array
    // of N elements equal, where
    // the first array element
    // equals M
    static int steps(int N, int M)
    {
        // Corner Case 1: When N = 1
        if (N == 1)
             return 0;

        // Corner Case 2: When N = 2
        else if (N == 2) // corner case 2
            return M;

        return 2 * M + (N - 3);
    }

    // Driver Code
    public static void Main ()
    {
        int N = 4, M = 4;
        Console.WriteLine(steps(N, M));
    }
}

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

Output:

```
9
```

Improved By : vt_m, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/make-array-elements-equal-in-minimum-steps/

Chapter 58

Making elements of two arrays same with minimum increment/decrement

Making elements of two arrays same with minimum increment/decrement - GeeksforGeeks

Given two arrays of same size, we need to convert the first array into another with minimum operations. In an operation, we can either increment or decrement an element by one. Note that orders of appearance of elements do not need to be same.

Here to convert one number into another we can add or subtract 1 from it.

Examples :

> **Input :** a = { 3, 1, 1 }, b = { 1, 2, 2 }
> **Output :** 2
> **Explanation :** Here we can increase any 1 into 2 by 1 operation and 3 to 2 in one decrement operation. So a[] becomes {2, 2, 1} which is a permutation of b[].
>
> **Input :** a = { 3, 1, 1 }, b = { 1, 1, 2 }
> **Output :** 1

Algorithm :
1. First sort both the arrays.
2. After sorting we will run a loop in which we compare the first and second array elements and calculate the required operation needed to make first array equal to second.

Below is implementation of the above approach

C++

```
 // CPP program to find minimum increment/decrement
// operations to make array elements same.
```

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

int MinOperation(int a[], int b[], int n)
{
    // sorting both arrays in
    // ascending order
    sort(a, a + n);
    sort(b, b + n);

    // variable to store the
    // final result
    int result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for (int i = 0; i < n; ++i) {
        if (a[i] > b[i])
            result = result + abs(a[i] - b[i]);

        else if (a[i] < b[i])
            result = result + abs(a[i] - b[i]);
    }

    return result;
}

// Driver code
int main()
{
    int a[] = { 3, 1, 1 };
    int b[] = { 1, 2, 2 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << MinOperation(a, b, n);
    return 0;
}
```

Java

```
 // Java program to find minimum
// increment/decrement operations
// to make array elements same.
```

```
import java.util.Arrays;
import java.io.*;

class GFG
{
static int MinOperation(int a[],
                        int b[],
                        int n)
{
    // sorting both arrays
    // in ascending order
    Arrays.sort(a);
    Arrays.sort(b);


    // variable to store
    // the final result
    int result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for (int i = 0; i < n; ++i)
    {
        if (a[i] > b[i])
            result = result +
                     Math.abs(a[i] - b[i]);

        else if (a[i] < b[i])
            result = result +
                     Math.abs(a[i] - b[i]);
    }

    return result;
}

// Driver code
public static void main (String[] args)
{
    int a[] = {3, 1, 1};
    int b[] = {1, 2, 2};
    int n = a.length;
    System.out.println(MinOperation(a, b, n));
```

```
}
}

// This code is contributed
// by akt_mit
```

PHP

```
 <?php
// PHP program to find minimum
// increment/decrement operations
// to make array elements same.
function MinOperation($a, $b, $n)
{
    // sorting both arrays in
    // ascending order

    sort($a);
    sort($b);

    // variable to store
    // the final result
    $result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for ($i = 0; $i < $n; ++$i)
    {
        if ($a[$i] > $b[$i])
            $result = $result + abs($a[$i] -
                                    $b[$i]);

        else if ($a[$i] < $b[$i])
            $result = $result + abs($a[$i] -
                                    $b[$i]);
    }

    return $result;
}

// Driver code
$a = array ( 3, 1, 1 );
```

```
$b = array ( 1, 2, 2 );
$n = sizeof($a);
echo MinOperation($a, $b, $n);

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

Output :

```
2
```

Time Complexity : O(n Log n)

Improved By : jit_t

Source

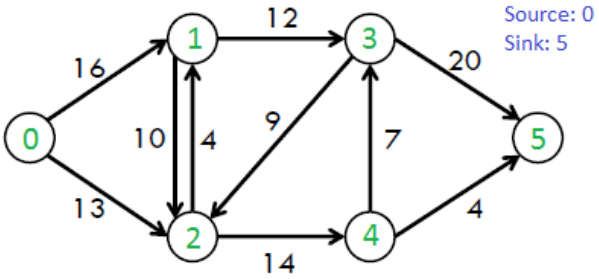
https://www.geeksforgeeks.org/making-elements-of-two-arrays-same-with-minimum-incrementdecrement/

Chapter 59

Max Flow Problem Introduction

Max Flow Problem Introduction - GeeksforGeeks

Maximum flow problems involve finding a feasible flow through a single-source, single-sink flow network that is maximum.
Let's take an image to explain how above definition wants to say.



Each edge is labeled with a capacity, the maximum amount of stuff that it can carry. The goal is to figure out how much stuff can be pushed from the vertex s(source) to the vertex t(sink).

maximum flow possible is : **23**

Following are different approaches to solve the problem :

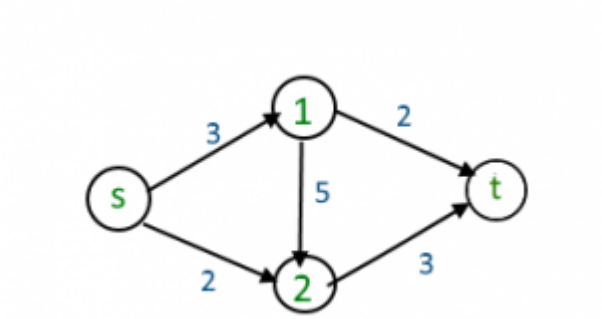
1. Naive Greedy Algorithm Approach (May not produce optimal or correct result)

Greedy approach to the maximum flow problem is to start with the all-zero flow and greedily produce flows with ever-higher value. The natural way to proceed from one to the next is to send more flow on some path from s to t
How Greedy approach work to find the maximum flow :

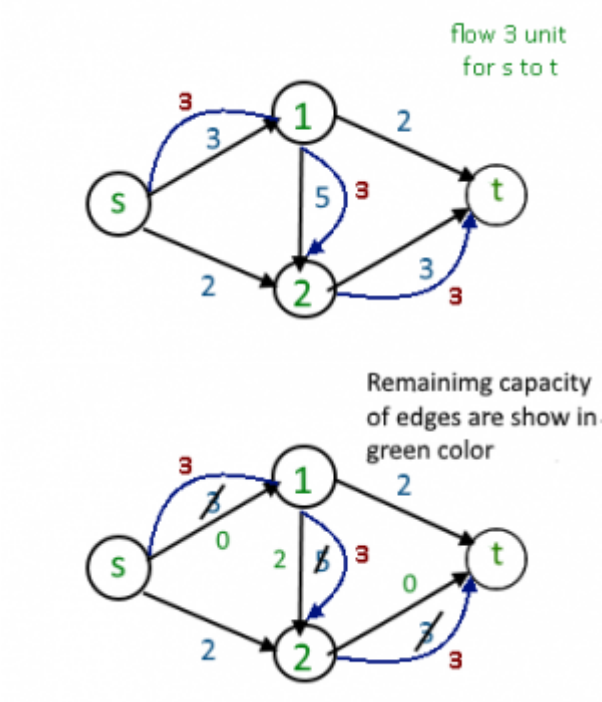
```
E number of edge
```

```
f(e) flow of edge
C(e) capacity of edge

1) Initialize : max_flow = 0
                f(e) = 0 for every edge 'e' in E

2) Repeat search for an s-t path P while it exists.
   a) Find if there is a path from s to t using BFS
      or DFS. A path exists if f(e) < C(e) for
      every edge e on the path.
   b) If no path found, return max_flow.
   c) Else find minimum edge value for path P

      // Our flow is limited by least remaining
      // capacity edge on path P.
      (i) flow = min(C(e)- f(e)) for path P ]
             max_flow += flow
      (ii) For all edge e of path increment flow
             f(e) += flow

3) Return max_flow
```

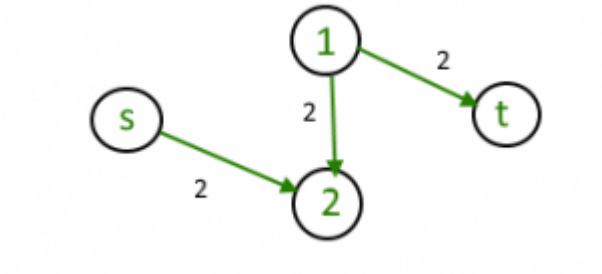
Note that the path search just needs to determine whether or not there is an s-t path in the subgraph of edges e with f(e) < C(e). This is easily done in linear time using BFS or DFS.



There is a path from source (s) to sink(t) [s -> 1 -> 2 -> t] with maximum flow 3 unit (path show in blue color)



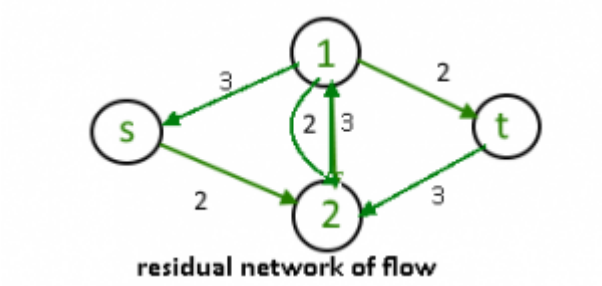
After removing all useless edge from graph it's look like



For above graph there is no path from source to sink so maximum flow : 3 unit But maximum flow is 5 unit. to over come form this issue we use residual Graph.

2. Residual Graphs

The idea is to extend the naive greedy algorithm by allowing "undo" operations. For example, from the point where this algorithm gets stuck in above image, we'd like to route two more units of flow along the edge (s, 2), then backward along the edge (1, 2), undoing 2 of the 3 units we routed the previous iteration, and finally along the edge (1,t)



backward edge : (f(e)) and forward edge : (C(e) – f(e))

We need a way of formally specifying the allowable “undo” operations. This motivates the following simple but important defnition, of a residual network. The idea is that, given a graph G and a flow f in it, we form a new flow network G_f that has the same vertex set of G and that has two edges for each edge of G. An edge e = (1,2) of G that carries ?ow f(e) and has capacity C(e) (for above image) spawns a “forward edge” of G_f with capacity C(e)-f(e) (the room remaining) and a “backward edge” (2,1) of G_f with capacity f(e) (the amount of previously routed ?ow that can be undone). source(s)- sink(t) paths with f(e) < C(e) for all edges, as searched for by the naive greedy algorithm, correspond to the special case of s-t paths of G_f that comprise only forward edges.

The idea of residual graph is used The Ford-Fulkerson and Dinic’s algorithms

Source :
http://theory.stanford.edu/~tim/w16/l/l1.pdf

Source

https://www.geeksforgeeks.org/max-flow-problem-introduction/

Chapter 60

Maximize array sum after K negations Set 1

Maximize array sum after K negations Set 1 - GeeksforGeeks

Given an array of size n and a number k. We must modify array K number of times. Here modify array means in each operation we can replace any array element arr[i] by -arr[i]. We need to perform this operation in such a way that after K operations, sum of array must be maximum?

Examples :

```
Input : arr[] = {-2, 0, 5, -1, 2}
        K = 4
Output: 10
// Replace (-2) by -(-2), array becomes {2, 0, 5, -1, 2}
// Replace (-1) by -(-1), array becomes {2, 0, 5, 1, 2}
// Replace (0) by -(0), array becomes {2, 0, 5, 1, 2}
// Replace (0) by -(0), array becomes {2, 0, 5, 1, 2}

Input : arr[] = {9, 8, 8, 5}
        K = 3
Output: 20
```

This problem has very **simple solution**, we just have to replace the minimum element arr[i] in array by -arr[i] for current operation. In this way we can make sum of array maximum after K operations. Once interesting case is, once minimum element becomes 0, we don't need to make any more changes.
C++

```
// C++ program to to maximize array sum after
```

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

// This function does k operations on array
// in a way that maximize the array sum.
// index --> stores the index of current minimum
//           element for j'th operation
int maximumSum(int arr[], int n, int k)
{
    // Modify array K number of times
    for (int i=1; i<=k; i++)
    {
        int min = INT_MAX;
        int index = -1;

        // Find minimum element in array for
        // current operation and modify it
        // i.e; arr[j] --> -arr[j]
        for (int j=0; j<n; j++)
        {
            if (arr[j] < min)
            {
                min = arr[j];
                index = j;
            }
        }

        // this the condition if we find 0 as
        // minimum element, so it will useless to
        // replace 0 by -(0) for remaining operations
        if (min == 0)
            break;

        // Modify element of array
        arr[index] = -arr[index];
    }

    // Calculate sum of array
    int sum = 0;
    for (int i=0; i<n; i++)
        sum += arr[i];
    return sum;
}

// Driver program to test the case
int main()
{
```

```
    int arr[] = {-2, 0, 5, -1, 2};
    int k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << maximumSum(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to to maximize array
// sum after k operations.

class GFG
{
    // This function does k operations
    // on array in a way that maximize
    // the array sum. index --> stores
    // the index of current minimum
    // element for j'th operation
    static int maximumSum(int arr[], int n, int k)
    {
        // Modify array K number of times
        for (int i = 1; i <= k; i++)
        {
            int min = +2147483647;
            int index = -1;

            // Find minimum element in array for
            // current operation and modify it
            // i.e; arr[j] --> -arr[j]
            for (int j=0; j<n; j++)
            {
                if (arr[j] < min)
                {
                    min = arr[j];
                    index = j;
                }
            }

            // this the condition if we find 0 as
            // minimum element, so it will useless to
            // replace 0 by -(0) for remaining operations
            if (min == 0)
                break;

            // Modify element of array
            arr[index] = -arr[index];
        }
```

```
        // Calculate sum of array
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += arr[i];
        return sum;
    }


    // Driver program
    public static void main(String arg[])
    {
    int arr[] = {-2, 0, 5, -1, 2};
        int k = 4;
        int n = arr.length;
        System.out.print(maximumSum(arr, n, k));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to to maximize
# array sum after k operations.

# This function does k operations on array
# in a way that maximize the array sum.
# index --> stores the index of current
# minimum element for j'th operation
def maximumSum(arr, n, k):

    # Modify array K number of times
    for i in range(1, k + 1):

        min = +2147483647
        index = -1

        # Find minimum element in array for
        # current operation and modify it
        # i.e; arr[j] --> -arr[j]
        for j in range(n):

            if (arr[j] < min):

                min = arr[j]
                index = j
```

```
        # this the condition if we find 0 as
        # minimum element, so it will useless to
        # replace 0 by -(0) for remaining operations
        if (min == 0):
            break

        # Modify element of array
        arr[index] = -arr[index]


    # Calculate sum of array
    sum = 0
    for i in range(n):
        sum += arr[i]
    return sum

# Driver program
arr = [-2, 0, 5, -1, 2]
k = 4
n = len(arr)
print(maximumSum(arr, n, k))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to to maximize array
// sum after k operations.
using System;

class GFG
{

    // This function does k operations
    // on array in a way that maximize
    // the array sum. index --> stores
    // the index of current minimum
    // element for j'th operation
    static int maximumSum(int []arr, int n,
                          int k)
    {

        // Modify array K number of times
        for (int i = 1; i <= k; i++)
        {
            int min = +2147483647;
            int index = -1;
```

```
            // Find minimum element in array for
            // current operation and modify it
            // i.e; arr[j] --> -arr[j]
            for (int j = 0; j < n; j++)
            {
                if (arr[j] < min)
                {
                    min = arr[j];
                    index = j;
                }
            }

            // this the condition if we find
            // 0 as minimum element, so it
            // will useless to replace 0 by -(0)
            // for remaining operations
            if (min == 0)
                break;

            // Modify element of array
            arr[index] = -arr[index];
        }

        // Calculate sum of array
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += arr[i];
        return sum;
    }

    // Driver code
    public static void Main()
    {
        int []arr = {-2, 0, 5, -1, 2};
        int k = 4;
        int n = arr.Length;
        Console.Write(maximumSum(arr, n, k));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to to maximize
// array sum after k operations.
```

```
// This function does k operations
// on array in a way that maximize
// the array sum. index --> stores
// the index of current minimum
// element for j'th operation
function maximumSum($arr, $n, $k)
{
    $INT_MAX = 0;
    // Modify array K
    // number of times
    for ($i = 1; $i <= $k; $i++)
    {
        $min = $INT_MAX;
        $index = -1;

        // Find minimum element in
        // array for current operation
        // and modify it i.e;
        // arr[j] --> -arr[j]
        for ($j = 0; $j < $n; $j++)
        {
            if ($arr[$j] < $min)
            {
                $min = $arr[$j];
                $index = $j;
            }
        }

        // this the condition if we
        // find 0 as minimum element,so
        // it will useless to replace 0
        // by -(0) for remaining operations
        if ($min == 0)
            break;

        // Modify element of array
        $arr[$index] = -$arr[$index];
    }

    // Calculate sum of array
    $sum = 0;
    for ($i = 0; $i < $n; $i++)
        $sum += $arr[$i];
    return $sum;
}

// Driver Code
$arr = array(-2, 0, 5, -1, 2);
```

```
$k = 4;
$n = sizeof($arr) / sizeof($arr[0]);
echo maximumSum($arr, $n, $k);

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

Output :

```
10
```

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

Maximize array sum after K negations Set 2

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/maximize-array-sun-after-k-negation-operations/

Chapter 61

Maximize array sum after K negations Set 2

Maximize array sum after K negations Set 2 - GeeksforGeeks

Given an array of size n and a number k. We must modify array K number of times. Here modify array means in each operation we can replace any array element arr[i] by -arr[i]. We need to perform this operation in such a way that after K operations, sum of array must be maximum?

Examples:

```
Input : arr[] = {-2, 0, 5, -1, 2}
        K = 4
Output: 10
// Replace (-2) by -(-2), array becomes {2, 0, 5, -1, 2}
// Replace (-1) by -(-1), array becomes {2, 0, 5, 1, 2}
// Replace (0) by -(0), array becomes {2, 0, 5, 1, 2}
// Replace (0) by -(0), array becomes {2, 0, 5, 1, 2}

Input : arr[] = {9, 8, 8, 5}
        K = 3
Output: 20
```

We have discussed a O(nk) solution in below post.

Maximize array sum after K negations Set 1

The idea used in above post is to replace the minimum element arr[i] in array by -arr[i] for current operation. In this way we can make sum of array maximum after K operations. Once interesting case is, once minimum element becomes 0, we don't need to make any more changes.

The implementation used in above solution uses linear search to find minimum element. The time complexity of the above discussed solution is O(nk)

In this post an optimized solution is implemented that uses a priority queue (or binary heap) to find minimum element quickly.

Below is Java implementation of the idea. It uses PriorityQueue class in Java.

```
 // A PriorityQueue based Java program to maximize array
// sum after k negations.
import java.util.*;

class maximizeSum
{
    public static int maxSum(int[] a, int k)
    {
        // Create a priority queue and insert all array elements
        // int
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for (int x : a)
            pq.add(x);

        // Do k negations by removing a minimum element k times
        while (k-- > 0)
        {
            // Retrieve and remove min element
            int temp = pq.poll();

            // Modify the minimum element and add back
            // to priority queue
            temp *= -1;
            pq.add(temp);
        }

        // Compute sum of all elements in priority queue.
        int sum = 0;
        for (int x : pq)
            sum += x;
        return sum;
    }

    // Driver code
    public static void main (String[] args)
    {
        int[] arr = {-2, 0, 5, -1, 2};
        int k = 4;
        System.out.println(maxSum(arr, k));
    }
}
```

Output:

```
10
```

Note that this optimized solution can be implemented in O(n + kLogn) time as we can create a priority queue (or binary heap) in O(n) time.

Source

https://www.geeksforgeeks.org/maximize-array-sum-k-negations-set-2/

Chapter 62

Maximize sum of consecutive differences in a circular array

Maximize sum of consecutive differences in a circular array - GeeksforGeeks

Given an array of **n** elements. Consider array as circular array i.e element after a_n is a_1. The task is to find maximum sum of the difference between consecutive elements with rearrangement of array element allowed i.e after rearrangement of element find $a_1 - a_2 + a_2 - a_3 + \ldots\ldots + a_{n-1} - a_n + a_n - a_1$.

Examples:

```
Input : arr[] = { 4, 2, 1, 8 }
Output : 18
Rearrange given array as : { 1, 8, 2, 4 }
Sum of difference between consecutive element
= |1 - 8| + |8 - 2| + |2 - 4| + |4 - 1|
= 7 + 6 + 2 + 3
= 18.

Input : arr[] = { 10, 12, 15 }
Output : 10
```

The idea is to use Greedy Approach and try to bring elements having greater difference closer.
Consider the sorted permutation of the given array a_1, a_1, a_2,...., a_{n-1}, a_n such that $a_1 < a_2 < a_3 \ldots. < a_{n-1} < a_n$.
Now, to obtain the answer having maximum sum of difference between consecutive element, arrange element in following manner:
a_1, a_n, a_2, a_{n-1},...., $a_{n/2}$, $a_{(n/2)+1}$
We can observe that the arrangement produces the optimal answer, as all a_1, a_2, a_3,....., $a_{(n/2)-1}$, $a_{n/2}$ are subtracted twice while $a_{(n/2)+1}$, $a_{(n/2)+2}$, $a_{(n/2)+3}$,....., a_{n-1}, a_n are added

twice.

C++

```
 // C++ program to maximize the sum of difference
// between consecutive elements in circular array
#include <bits/stdc++.h>
using namespace std;

// Return the maximum Sum of difference between
// consecutive elements.
int maxSum(int arr[], int n)
{
    int sum = 0;

    // Sorting the array.
    sort(arr, arr + n);

    // Subtracting a1, a2, a3,....., a(n/2)-1, an/2
    // twice and adding a(n/2)+1, a(n/2)+2, a(n/2)+3,.
    // ...., an - 1, an twice.
    for (int i = 0; i < n/2; i++)
    {
        sum -= (2 * arr[i]);
        sum += (2 * arr[n - i - 1]);
    }

    return sum;
}

// Driver Program
int main()
{
    int arr[] = { 4, 2, 1, 8 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << maxSum(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to maximize the sum of difference
// between consecutive elements in circular array
import java.io.*;
import java.util.Arrays;

class MaxSum
{
```

```
    // Return the maximum Sum of difference between
    // consecutive elements.
    static int maxSum(int arr[], int n)
    {
        int sum = 0;

        // Sorting the array.
        Arrays.sort(arr);

        // Subtracting a1, a2, a3,....., a(n/2)-1,
        // an/2 twice and adding a(n/2)+1, a(n/2)+2,
        // a(n/2)+3,....., an - 1, an twice.
        for (int i = 0; i < n/2; i++)
        {
            sum -= (2 * arr[i]);
            sum += (2 * arr[n - i - 1]);
        }

        return sum;
    }

    // Driver Program
    public static void main (String[] args)
    {
        int arr[] = { 4, 2, 1, 8 };
        int n = arr.length;
        System.out.println(maxSum(arr, n));
    }
}
/*This code is contributed by Prakriti Gupta*/
```

Python3

```
 # Python3 program to maximize the sum of difference
# between consecutive elements in circular array

# Return the maximum Sum of difference
# between consecutive elements
def maxSum(arr, n):
    sum = 0

    # Sorting the array
    arr.sort()

    # Subtracting a1, a2, a3,....., a(n/2)-1, an/2
    # twice and adding a(n/2)+1, a(n/2)+2, a(n/2)+3,.
    # ...., an - 1, an twice.
    for i in range(0, int(n / 2)) :
```

```
        sum -= (2 * arr[i])
        sum += (2 * arr[n - i - 1])

    return sum


# Driver Program
arr = [4, 2, 1, 8]
n = len(arr)
print (maxSum(arr, n))

# This code is contributed by Shreyanshi Arun.
```

C#

```
 // C# program to maximize the sum of difference
// between consecutive elements in circular array
using System;

class MaxSum {

    // Return the maximum Sum of difference
    // between consecutive elements.
    static int maxSum(int[] arr, int n)
    {
        int sum = 0;

        // Sorting the array.
        Array.Sort(arr);

        // Subtracting a1, a2, a3, ....., a(n/2)-1,
        // an/2 twice and adding a(n/2)+1, a(n/2)+2,
        // a(n/2)+3, ....., an - 1, an twice.
        for (int i = 0; i < n / 2; i++) {
            sum -= (2 * arr[i]);
            sum += (2 * arr[n - i - 1]);
        }

        return sum;
    }

    // Driver Program
    public static void Main()
    {
        int[] arr = { 4, 2, 1, 8 };
        int n = arr.Length;
        Console.WriteLine(maxSum(arr, n));
    }
```

```
}

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

Output :

```
18
```

Time Complexity: O(nlogn).
Auxiliary Space : O(1)

Source

https://www.geeksforgeeks.org/maximize-sum-consecutive-differences-circular-array/

Chapter 63

Maximize the profit by selling at-most M products

Maximize the profit by selling at-most M products - GeeksforGeeks

Given two lists that contains cost prices CP[] and selling prices SP[] of products respectively. The task is to maximize the profit by selling at-most 'M' prodcts.

Examples:

> **Input:** N = 5, M = 3
> CP[]= {5, 10, 35, 7, 23}
> SP[] = {11, 10, 0, 9, 19}
> **Output:** 8
> Profit on 0th product i.e. 11-5 = 6
> Profit on 3rd product i.e. 9-7 = 2
> Selling any other product will not give profit.
> So, total profit = 6+2 = 8.
>
> **Input:** N = 4, M = 2
> CP[] = {17, 9, 8, 20}
> SP[] = {10, 9, 8, 27}
> **Output:** 7

Approach:

1. Store the profit/loss on buying and selling of each product i.e. **SP[i]-CP[i]** in an array.
2. Sort that array in descending order.
3. Add the positive values up to M values as positive values denote profit.
4. Return Sum.

Below is the implementation of above approach:
C++

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

// Function to find profit
int solve(int N, int M, int cp[], int sp[])
{
    int profit[N];

    // Calculating profit for each gadget
    for (int i = 0; i < N; i++)
        profit[i] = sp[i] - cp[i];

    // sort the profit array in decending order
    sort(profit, profit + N, greater<int>());

    // variable to calculate total profit
    int sum = 0;

    // check for best M profits
    for (int i = 0; i < M; i++) {
        if (profit[i] > 0)
            sum += profit[i];
        else
            break;
    }

    return sum;
}

// Driver Code
int main()
{

    int N = 5, M = 3;
    int CP[] = { 5, 10, 35, 7, 23 };
    int SP[] = { 11, 10, 0, 9, 19 };

    cout << solve(N, M, CP, SP);

    return 0;
}
```

Java

```
 // Java implementation of above approach:
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{

// Function to find profit
static int solve(int N, int M,
                 int cp[], int sp[])
{
    Integer []profit = new Integer[N];

    // Calculating profit for each gadget
    for (int i = 0; i < N; i++)
        profit[i] = sp[i] - cp[i];

    // sort the profit array
    // in decending order
    Arrays.sort(profit, Collections.reverseOrder());

    // variable to calculate total profit
    int sum = 0;

    // check for best M profits
    for (int i = 0; i < M; i++)
    {
        if (profit[i] > 0)
            sum += profit[i];
        else
            break;
    }

    return sum;
}

// Driver Code
public static void main(String args[])
{
    int N = 5, M = 3;
    int CP[] = { 5, 10, 35, 7, 23 };
    int SP[] = { 11, 10, 0, 9, 19 };

    System.out.println(solve(N, M, CP, SP));
}
}
```

```
// This code is contributed
// by Subhadeep Gupta
```

Python3

```
 # Python3 implementation
# of above approach

# Function to find profit
def solve(N, M, cp, sp) :

    # take empty list
    profit = []

    # Calculating profit
    # for each gadget
    for i in range(N) :
        profit.append(sp[i] - cp[i])

    # sort the profit array
    # in decending order
    profit.sort(reverse = True)

    sum = 0

    # check for best M profits
    for i in range(M) :
        if profit[i] > 0 :
            sum += profit[i]
        else :
            break

    return sum

# Driver Code
if __name__ == "__main__" :

    N, M = 5, 3
    CP = [5, 10, 35, 7, 23]
    SP = [11, 10, 0, 9, 19]

    # function calling
    print(solve(N, M, CP, SP))

# This code is contributed
# by ANKITRAI1
```

C#

```
// C# implementation of above approach:
using System;

class GFG
{

// Function to find profit
static int solve(int N, int M,
int[] cp, int[] sp)
{
int[] profit = new int[N];

// Calculating profit for each gadget
for (int i = 0; i < N; i++) profit[i] = sp[i] - cp[i]; // sort the profit array // in descending
order Array.Sort(profit); Array.Reverse(profit); // variable to calculate total profit int sum
= 0; // check for best M profits for (int i = 0; i < M; i++) { if (profit[i] > 0)
sum += profit[i];
else
break;
}

return sum;
}

// Driver Code
public static void Main()
{
int N = 5, M = 3;
int[] CP = { 5, 10, 35, 7, 23 };
int[] SP = { 11, 10, 0, 9, 19 };

Console.Write(solve(N, M, CP, SP));
}
}

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

PHP

```
0)
$sum += $profit[$i];
else
break;
}

return $sum;
}

// Driver Code
$N = 5;
$M = 3;
$CP = array( 5, 10, 35, 7, 23 );
$SP = array( 11, 10, 0, 9, 19 );
```

```
echo solve($N, $M, $CP, $SP);

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

Output:

```
8
```

Improved By : ANKITRAI1, tufan_gupta2000, ChitraNayal

Source

https://www.geeksforgeeks.org/maximize-the-profit-by-selling-at-most-m-products/

Chapter 64

Maximize the sum of X+Y elements by picking X and Y elements from 1st and 2nd array

Maximize the sum of X+Y elements by picking X and Y elements from 1st and 2nd array - GeeksforGeeks

Given two arrays of size N, and two numbers X and Y, the task is to maximize the sum by considering the below points:

- Pick **x** values from the first array and **y** values from the second array such that the sum of X+Y values is maximum.
- It is given that X + Y is equal to N.

Examples:

> **Input:** arr1[] = {1, 4, 1}, arr2[] = {2, 5, 3}, N = 3, X = 2, Y = 1
> **Output:** 8
> In order to maximize sum from 2 arrays,
> pick 1st and 2nd element from first array and 3rd from second array.
>
> **Input:** A[] = {1, 4, 1, 2}, B[] = {4, 3, 2, 5}, N = 4, X = 2, Y = 2
> **Output:** 14

Approach: A greedy approach can be used to solve the above problem. Below are the required steps:

- Find those elements of arrays first that have maximum value by finding the highest difference between elements of two arrays.

- For that, find the absolute difference between the value of the first and second array and then store it in some another array.
- Sort this array in decreasing order.
- While sorting, track the original positions of elements in the arrays.
- Now compare the elements of the two arrays and add the greater value to the maxAmount.
- If both have the same value, add an element of the first array if X is not zero else add an element of the second array.
- After traversing the arrays completely return the maxAmount calculated.

Below is the implementation of above approach :

```
 // C++ program to print the maximum
// possible sum from two arrays.
#include <bits/stdc++.h>
using namespace std;

// class that store values of two arrays
// and also store their absolute difference
class triplet {
public:
    int first;
    int second;
    int diff;
    triplet(int f, int s, int d)
        : first(f), second(s), diff(d)
    {
    }
};

// Compare function used to sort array in decreasing order
bool compare(triplet& a, triplet& b)
{
    return a.diff > b.diff; // decreasing order
}

/// Function to find the maximum possible
/// sum that can be generated from 2 arrays
int findMaxAmount(int arr1[], int arr2[], int n, int x, int y)
{
    // vector where each index stores 3 things:
    // Value of 1st array
    // Value of 2nd array
    // Their absolute difference
```

```
vector<triplet> v;

for (int i = 0; i < n; i++) {
    triplet t(arr1[i], arr2[i], abs(arr1[i] - arr2[i]));
    v.push_back(t);
}

// sort according to their absolute difference
sort(v.begin(), v.end(), compare);

// it will store maximum sum
int maxAmount = 0;

int i = 0;

// Run loop for N times or
// value of X or Y becomes zero
while (i < n && x > 0 && y > 0) {

    // if 1st array element has greater
    // value, add it to maxAmount
    if (v[i].first > v[i].second) {
        maxAmount += v[i].first;
        x--;
    }

    // if 2nd array element has greater
    // value, add it to maxAmount
    if (v[i].first < v[i].second) {
        maxAmount += v[i].second;
        y--;
    }

    // if both have same value, add element
    // of first array if X is not zero
    // else add element of second array
    if (v[i].first == v[i].second) {
        if (x > 0) {
            maxAmount += v[i].first;
            x--;
        }
        else if (y > 0) {
            maxAmount += v[i].second;
            y--;
        }
    }

    // increment after picking element
```

```
        i++;
    }

    // add the remaining values
    // of first array to maxAmount
    while (i < v.size() && x--) {
        maxAmount += v[i++].first;
    }

    // add the remaining values of
    // second array to maxAmount
    while (i < v.size() && y--) {
        maxAmount += v[i++].second;
    }

    return maxAmount;
}

// Driver Code
int main()
{
    int A[] = { 1, 4, 1, 2 };
    int B[] = { 4, 3, 2, 5 };
    int n = sizeof(A) / sizeof(A[0]);

    int X = 2, Y = 2;

    cout << findMaxAmount(A, B, n, X, Y) << "\n";
}
```

Output:

```
14
```

Time complexity: O(N log N)

Source

https://www.geeksforgeeks.org/maximize-the-sum-of-xy-elements-by-picking-x-and-y-elements-from-1st-and-2nd-a

Chapter 65

Maximize the sum of arr[i]*i

Maximize the sum of arr[i]*i - GeeksforGeeks

Given an array of **N** integers. You are allowed to rearrange the element of the array. The task is to find the maximum value of Σarr[i]*i, where i = 0, 1, 2,...., n – 1.

Examples:

```
Input : N = 4, arr[] = { 3, 5, 6, 1 }
Output : 31
If we arrange arr[] as { 1, 3, 5, 6 }.
Sum of arr[i]*i is 1*0 + 3*1 + 5*2 + 6*3
= 31, which is maximum

Input : N = 2, arr[] = { 19, 20 }
Output : 20
```

A **simple solution** is to generate all permutations of given array. For every permutation, compute the value of Σarr[i]*i and finally return the maximum value.

An **efficient solution** is based on the fact that the largest value should be scaled maximum and smallest value should be scaled minimum. So we multiply minimum value of i with minimum value of arr[i]. So, sort the given array in increasing order and compute the sum of ari]*i, where i = 0 to n-1.

Below is the implementation of this approach:
C++

```
 // CPP program to find the maximum value
// of i*arr[i]
#include<bits/stdc++.h>
using namespace std;
```

```
int maxSum(int arr[], int n)
{
  // Sort the array
  sort(arr, arr + n);

  // Finding the sum of arr[i]*i
  int sum = 0;
  for (int i = 0; i < n; i++)
    sum += (arr[i]*i);

  return sum;
}

// Driven Program
int main()
{
  int arr[] = { 3, 5, 6, 1 };
  int n = sizeof(arr)/sizeof(arr[0]);

  cout << maxSum(arr, n) << endl;
  return 0;
}
```

Java

```
 // Java program to find the
// maximum value of i*arr[i]
import java.util.*;

class GFG {

    static int maxSum(int arr[], int n)
    {
    // Sort the array
    Arrays.sort(arr);

    // Finding the sum of arr[i]*i
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += (arr[i] * i);

    return sum;
    }

    // Driven Program
    public static void main(String[] args)
    {
    int arr[] = { 3, 5, 6, 1 };
```

```
    int n = arr.length;

    System.out.println(maxSum(arr, n));

    }
}
// This code is contributed by Prerna Saini
```

Python3

```
 # Python program to find the
# maximum value of i*arr[i]
def maxSum(arr,n):

    #  Sort the array
    arr.sort()

    # Finding the sum of
    # arr[i]*i
    sum = 0
    for i in range(n):
        sum += arr[i] * i

    return sum

# Driver Program
arr = [3,5,6,1]
n = len(arr)
print(maxSum(arr,n))

# This code is contributed
# by Shrikant13
```

C#

```
 // C# program to find the
// maximum value of i*arr[i]
using System;

class GFG {

    // Function to find the
    // maximum value of i*arr[i]
    static int maxSum(int[] arr, int n)
    {

        // Sort the array
```

```
        Array.Sort(arr);

        // Finding the sum of arr[i]*i
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += (arr[i] * i);

        return sum;
    }

    // Driver code
    static public void Main()
    {
        int[] arr = {3, 5, 6, 1};
        int n = arr.Length;

        Console.WriteLine(maxSum(arr, n));

    }
}

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

PHP

```
 <?php
// PHP program to find the
// maximum value of i*arr[i]

// function returns the
// maximum value of i*arr[i]
function maxSum($arr, $n)
{
    // Sort the array
    sort($arr);

    // Finding the sum
    // of arr[i]*i
    $sum = 0;
    for ($i = 0; $i < $n; $i++)
        $sum += ($arr[$i] * $i);

    return $sum;
}

// Driver Code
$arr = array( 3, 5, 6, 1 );
$n = count($arr);
```

```
echo maxSum($arr, $n);

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

Output:

```
31
```

Time Complexity : O(n Log n)

Improved By : shrikanth13, jit_t, vt_m

Source

https://www.geeksforgeeks.org/maximize-sum-arrii/

Chapter 66

Maximum Length Chain of Pairs DP-20

Maximum Length Chain of Pairs DP-20 - GeeksforGeeks

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.
C

```
 #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;
}
```

Java

```
 class Pair{
    int a;
    int b;
```

```
    public Pair(int a, int b) {
        this.a = a;
        this.b = b;
    }

    // This function assumes that arr[] is sorted in increasing order
    // according the first (or smaller) values in pairs.
    static int maxChainLength(Pair arr[], int n)
    {
       int i, j, max = 0;
       int mcl[] = new 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];

       return max;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        Pair arr[] = new Pair[] {new Pair(5,24), new Pair(15, 25),
                                 new Pair (27, 40), new Pair(50, 60)};
        System.out.println("Length of maximum size chain is " +
                                 maxChainLength(arr, arr.length));
    }
}
```

Python3

```
 class Pair(object):
    def __init__(self, a, b):
        self.a = a
        self.b = b
```

```
# This function assumes that arr[] is sorted in increasing
# order according the first (or smaller) values in pairs.
def maxChainLength(arr, n):

    max = 0

    # Initialize MCL(max chain length) values for all indices
    mcl = [1 for i in range(n)]

    # Compute optimized chain length values in bottom up manner
    for i in range(1, n):
        for j in range(0, i):
            if (arr[i].a > arr[j].b and 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 in range(n):
        if (max < mcl[i]):
            max = mcl[i]

    return max

# Driver program to test above function
arr = [Pair(5, 24), Pair(15, 25), Pair(27, 40), Pair(50, 60)]

print('Length of maximum size chain is',
      maxChainLength(arr, len(arr)))

# This code is contributed by Soumen Ghosh
```

C#

```
 // Dynamic C# program to find
// Maximum Length Chain of Pairs
using System;

class Pair {
    int a;
    int b;

    public Pair(int a, int b)
    {
        this.a = a;
        this.b = b;
```

```
    }

    // This function assumes that arr[]
    // is sorted in increasing order
    // according the first (or smaller)
    // values in pairs.
    static int maxChainLength(Pair []arr, int n)
    {
        int i, j, max = 0;
        int []mcl = new 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] now stores the maximum
          // chain length ending with pair i
          mcl[i] = mcl[j] + 1;

        // Pick maximum of all MCL values
        for ( i = 0; i < n; i++ )
            if (max < mcl[i] )
                max = mcl[i];

        return max;
    }

    // Driver Code
    public static void Main()
    {
        Pair []arr = new Pair[] {new Pair(5,24), new Pair(15, 25),
                                 new Pair (27, 40), new Pair(50, 60)};
        Console.Write("Length of maximum size chain is " +
                      maxChainLength(arr, arr.Length));
    }
}

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

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 problemand can be solved in (nLogn) 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.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/maximum-length-chain-of-pairs-dp-20/

Chapter 67

Maximum elements that can be made equal with k updates

Maximum elements that can be made equal with k updates - GeeksforGeeks

Given an array and a value k. We have to find the maximum number of equal elements possible for the array so that we can increase the elements of the array by incrementing a total of at-most k.

Examples:

> **Input :** array = { 2, 4, 9 }, k = 3
> **Output :** 2
> We are allowed to do at most three increments. We can make two elements 4 by increasing 2 by 2. Note that we can not make two elements 9 as converting 4 to 9 requires 5 increments.
>
> **Input :** array = { 5, 5, 3, 1 }, k = 5
> **Output :** 3
> **Explanation:** Here 1st and 2nd elements are equal. Then we can increase 3rd element 3 upto 5. Then k becomes (k-2) = 3. Now we can't increase 1 to 5 because k value is 3 and we need 4 for the updation. Thus equal elements possible are 3. Here we can also increase 1 to 5. Then also we have 3 because we can't update 3 to 5.
>
> **Input :** array = { 5, 5, 3, 1 }, k = 6
> **Output :** 4

Naive Approach: In the naive approach we have an algorithm in **O(n^2)** time in which we check for each element how many other elements can be incremented so that they will become equal to them.

Efficient Approach: In this approach, first we will sort the array. Then we maintain two arrays. First is prefix sum array which stores the prefix sum of the array and another is maxx[] array which stores the maximum element found till every point, i.e., max[i] means

maximum element from 1 to i. After storing these values in prefix[] array and maxx[] array, we do the binary search from 1 to n(number of elements of the array) to calculate how many elements which can be incremented to make them equal. In the binary search, we use one function in which we determine *what is the number of elements can be incremented to make them equal to a single value.*

C++

```
 // C++ program to find maximum elements that can
// be made equal with k updates
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the maximum number of
// equal elements possible with atmost K increment
// of values .Here we have done sliding window
// to determine that whether there are x number of
// elements present which on increment will become
// equal. The loop here will run in fashion like
// 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
bool ElementsCalculationFunc(int pre[], int maxx[],
                             int x, int k, int n)
{
    for (int i = 0, j = x; j <= n; j++, i++) {

        // It can be explained with the reasoning
        // that if for some x number of elements
        // we can update the values then the
        // increment to the segment (i to j having
        // length -> x) so that all will be equal is
        // (x*maxx[j]) this is the total sum of
        // segment and (pre[j]-pre[i]) is present sum
        // So difference of them should be less than k
        // if yes, then that segment length(x) can be
        // possible return true
        if (x * maxx[j] - (pre[j] - pre[i]) <= k)
            return true;
    }
    return false;
}

void MaxNumberOfElements(int a[], int n, int k)
{
    // sort the array in ascending order
    sort(a, a + n);
    int pre[n + 1]; // prefix sum array
    int maxx[n + 1]; // maximum value array

    // Initializing the prefix array
```

```
    // and maximum array
    for (int i = 0; i <= n; ++i) {
        pre[i] = 0;
        maxx[i] = 0;
    }

    // set the first element of both
    // array
    maxx[0] = a[0];
    pre[0] = a[0];
    for (int i = 1; i < n; i++) {

        // Calculating prefix sum of the array
        pre[i] = pre[i - 1] + a[i];

        // Calculating max value upto that position
        // in the array
        maxx[i] = max(maxx[i - 1], a[i]);
    }

    // Binary seach applied for
    // computation here
    int l = 1, r = n, ans;
    while (l < r) {
        int mid = (l + r) / 2;

        if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n)) {
            ans = mid;
            l = mid + 1;
        }
        else
            r = mid - 1;
    }

    // printing result
    cout << ans << "\n";
}

int main()
{
    int arr[] = { 2, 4, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    MaxNumberOfElements(arr, n, k);
    return 0;
}
```

Java

```
 // java program to find maximum elements that can
// be made equal with k updates

import java.util.Arrays;
public class GFG {

    // Function to calculate the maximum number of
    // equal elements possible with atmost K increment
    // of values .Here we have done sliding window
    // to determine that whether there are x number of
    // elements present which on increment will become
    // equal. The loop here will run in fashion like
    // 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
    static boolean ElementsCalculationFunc(int pre[],
                      int maxx[], int x, int k, int n)
    {
        for (int i = 0, j = x; j <= n; j++, i++) {

            // It can be explained with the reasoning
            // that if for some x number of elements
            // we can update the values then the
            // increment to the segment (i to j having
            // length -> x) so that all will be equal is
            // (x*maxx[j]) this is the total sum of
            // segment and (pre[j]-pre[i]) is present sum
            // So difference of them should be less than k
            // if yes, then that segment length(x) can be
            // possible return true
            if (x * maxx[j] - (pre[j] - pre[i]) <= k)
                return true;
        }
        return false;
    }

    static void MaxNumberOfElements(int a[], int n, int k)
    {
        // sort the array in ascending order
        Arrays.sort(a);
        int []pre = new int[n + 1]; // prefix sum array
        int []maxx = new int[n + 1]; // maximum value array

        // Initializing the prefix array
        // and maximum array
        for (int i = 0; i <= n; ++i) {
            pre[i] = 0;
            maxx[i] = 0;
```

```
        }

        // set the first element of both
        // array
        maxx[0] = a[0];
        pre[0] = a[0];
        for (int i = 1; i < n; i++) {

            // Calculating prefix sum of the array
            pre[i] = pre[i - 1] + a[i];

            // Calculating max value upto that position
            // in the array
            maxx[i] = Math.max(maxx[i - 1], a[i]);
        }

        // Binary seach applied for
        // computation here
        int l = 1, r = n, ans=0;
        while (l < r) {

            int mid = (l + r) / 2;

            if (ElementsCalculationFunc(pre, maxx,
                                    mid - 1, k, n))
            {
                ans = mid;
                l = mid + 1;
            }
            else
                r = mid - 1;
        }

        // printing result
        System.out.print((int)ans + "\n");
    }

    public static void main(String args[]) {

        int arr[] = { 2, 4, 9 };
        int n = arr.length;
        int k = 3;

        MaxNumberOfElements(arr, n, k);

    }
}
```

```
// This code is contributed by Sam007
```

C#

```
 // C# program to find maximum elements that can
// be made equal with k updates
using System;

class GFG {

    // Function to calculate the maximum number of
    // equal elements possible with atmost K increment
    // of values .Here we have done sliding window
    // to determine that whether there are x number of
    // elements present which on increment will become
    // equal. The loop here will run in fashion like
    // 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
    static bool ElementsCalculationFunc(int []pre,
                      int []maxx, int x, int k, int n)
    {
        for (int i = 0, j = x; j <= n; j++, i++) {

            // It can be explained with the reasoning
            // that if for some x number of elements
            // we can update the values then the
            // increment to the segment (i to j having
            // length -> x) so that all will be equal is
            // (x*maxx[j]) this is the total sum of
            // segment and (pre[j]-pre[i]) is present sum
            // So difference of them should be less than k
            // if yes, then that segment length(x) can be
            // possible return true
            if (x * maxx[j] - (pre[j] - pre[i]) <= k)
                return true;
        }
        return false;
    }

    static void MaxNumberOfElements(int []a, int n, int k)
    {
        // sort the array in ascending order
        Array.Sort(a);
        int []pre = new int[n + 1]; // prefix sum array
        int []maxx = new int[n + 1]; // maximum value array

        // Initializing the prefix array
        // and maximum array
        for (int i = 0; i <= n; ++i) {
```

```
        pre[i] = 0;
        maxx[i] = 0;
    }

    // set the first element of both
    // array
    maxx[0] = a[0];
    pre[0] = a[0];
    for (int i = 1; i < n; i++) {

        // Calculating prefix sum of the array
        pre[i] = pre[i - 1] + a[i];

        // Calculating max value upto that position
        // in the array
        maxx[i] = Math.Max(maxx[i - 1], a[i]);
    }

    // Binary seach applied for
    // computation here
    int l = 1, r = n, ans=0;
    while (l < r) {

        int mid = (l + r) / 2;

        if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n))
        {
            ans = mid;
            l = mid + 1;
        }
        else
            r = mid - 1;
    }

    // printing result
    Console.Write ((int)ans + "\n");
}

// Driver code
public static void Main()
{
    int []arr = { 2, 4, 9 };
    int n = arr.Length;
    int k = 3;

    MaxNumberOfElements(arr, n, k);
}
```

```
}

// This code is contributed by Sam007
```

Output:

```
2
```

Time Complexity : **O(nlog(n))**
Space Complexity : **O(n)**

Improved By : Sam007

Source

https://www.geeksforgeeks.org/maximum-elements-can-made-equal-k-updates/

Chapter 68

Maximum number by concatenating every element in a rotation of an array

Maximum number by concatenating every element in a rotation of an array - GeeksforGeeks

Given an array of N elements. The task is to print the maximum number by concatenating every element in each rotation. In every rotation, the first element will take place of the last element in each rotation and vice versa.

Examples:

> **Input:** a[]: {54, 546, 548, 60}
> **Output:** 6054546548
> 1st Rotation: 5465486054
> 2nd Rotation: 5486054546
> 3rd Rotation: 6054546548
> 4th Rotation: 5454654860
>
> **Input:** a[]: {1, 4, 18, 96}
> **Output:** 961418

Approach: On observing carefully, it is found that the number which has the largest left-most digit in all elements will be the first element in the number. Since the concatenation has to be done in term of rotation of arrays. Concatenate all the numbers from the largest left most digit index to the end and then concatenate the elements from 0^{th} index to the largest left most digit index.

Below is the implementation of the above approach:

C++

```
// C++ program to print the
```

```
// Maximum number by concatenating
// every element in rotation of array
#include <bits/stdc++.h>
using namespace std;

// Function to print the largest number
void printLargest(int a[], int n)
{

    // store the index of largest
    // left most digit of elements
    int max=-1;
    int ind = -1;

    // Iterate for all numbers
    for(int i = 0 ;i<n;i++)
    {

        int num = a[i];

        // check for the the last digit
        while(num)
        {
            int r = num%10;
            num = num/10;
            if(num==0)
            {
                // check for the largest left most digit
                if(max<r)
                {
                    r=max;
                    ind = i;
                }
            }
        }
    }

    // print the largest number

    // print the rotation of array
    for(int i = ind;i<n;i++)
       cout << a[i];

    // print the rotation of array
    for(int i=0;i<ind;i++)
       cout << a[i];
}
```

```
// Driver Code
int main() {
    int a[] = {54, 546, 548, 60};
    int n = sizeof(a)/sizeof(a[0]);
    printLargest(a,n);
    return 0;
}
```

Java

```
 // Java program to print the
// Maximum number by concatenating
// every element in rotation of array
import java.util.*;
import java.lang.*;

public class GFG {
    // Function to print the largest number
    static void printLargest(int a[], int n)
    {
        // store the index of largest
        // left most digit of elements
        int max = -1;
        int ind = -1;

        // Iterate for all numbers
        for(int i = 0; i < n; i++)
        {
            int num = a[i];

            // check for the the last digit
            while(num > 0)
            {
                int r = num % 10;
                num = num/10;
                if(num == 0)
                {
                    // check for the largest left most digit
                    if(max < r)
                    {
                        r = max;
                        ind = i;
                    }
                }
            }
        }
        // print the largest number
```

```
        // print the rotation of array
        for(int i = ind; i < n; i++)
            System.out.print(a[i]);

        // print the rotation of array
        for(int i = 0; i < ind; i++)
            System.out.print(a[i]);
    }

    // Driver Code
    public static void main(String args[]) {
        int a[] = { 54, 546, 548, 60 };
        int n = a.length;
        printLargest(a, n);
    }
}
```

Python3

```
 # Python program to print the
# Maximum number by concatenating
# every element in rotation of array

# Function to print the largest number
def printLargest(a, n):


    # store the index of largest
    # left most digit of elements
    max=-1
    ind=-1

    # Iterate for all numbers
    for i in range(0,n):
         num = a[i]

        # check for the the last digit
         while(num):

            r = num%10;
            num = num/10;
            if(num==0):
                # check for the largest left most digit
                if(max<r):
                    r=max
                    ind = i;
```

```
    #print the largest number

    # print the rotation of array
    for i in range(ind,n):
        print(a[i],end=''),

    # print the rotation of array
    for i in range(0,ind) :
        print(a[i],end='')

# Driver Code
if __name__ == "__main__":
    a = [54, 546, 548, 60]
    n = len(a)
    printLargest(a,n)


# This code is contributed by Shivi_Aggarwal
```

PHP

```
 <?php
// PHP program to print
// the Maximum number by
// concatenating every
// element in rotation of array

// Function to print
// the largest number
function printLargest($a, $n)
{

    // store the index of largest
    // left most digit of elements
    $max = -1;
    $ind = -1;

    // Iterate for
    // all numbers
    for($i = 0 ; $i < $n; $i++)
    {

        $num = $a[$i];

        // check for the
```

```
        // the last digit
        while($num)
        {
            $r = $num % 10;
            $num = (int)$num / 10;
            if($num == 0)
            {
                // check for the largest
                // left most digit
                if($max < $r)
                {
                    $r = $max;
                    $ind = $i;
                }
            }
        }
    }

    // print the largest number

    // print the
    // rotation of array
    for($i = $ind; $i < $n; $i++)
    echo $a[$i];

    // print the
    // rotation of array
    for($i = 0; $i < $ind; $i++)
    echo $a[$i];
}

// Driver Code
$a = array (54, 546,
            548, 60);
$n = sizeof($a);
printLargest($a, $n);

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

Output:

```
6054546548
```

Improved By : jit_t, Nishant Tanwar, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/maximum-number-by-concatenating-every-element-in-a-rotation-of-an-array/

Chapter 69

Maximum number of customers that can be satisfied with given quantity

Maximum number of customers that can be satisfied with given quantity - GeeksforGeeks

A new variety of rice has been brought in supermarket and being available for the first time, the quantity of this rice is limited. Each customer demands the rice in two different packaging of size a and size b. The sizes a and b are decided by staff as per the demand.
Given the size of the packets a and b, the total quantity of rice available d and the number of customers n, find out maximum number of customers that can be satisfied with the given quantity of rice.
Display the total number of customers that can be satisfied and the index of customers that can be satisfied.

Note: If a customer orders 2 3, he requires 2 packets of size a and 3 packets of size b. Assume indexing of customers starts from 1.

Input:
The first line of input contains two integers n and d; next line contains two integers a and b. Next n lines contain two integers for each customer denoting total number of bags of size a and size b that customer requires.

Output:
Print maximum number of customers that can be satisfied and in next line print the space separated indexes of satisfied customers.

Examples:

```
Input : n = 5, d = 5
        a = 1, b = 1
        2 0
```

```
        3 2
        4 4
        10 0
        0 1
Output : 2
         5 1

Input : n = 6, d = 1000000000
       a = 9999, b = 10000
       10000 9998
       10000 10000
       10000 10000
       70000 70000
       10000 10000
       10000 10000
Output : 5
         1 2 3 5 6
```

Explanation:
In first example, the order of customers according to their demand is:

Customer ID	Demand
5	1
1	2
2	5
3	8
4	10

From this, it can easily be concluded that only customer 5 and customer 1 can be satisfied for total demand of $1 + 2 = 3$. Rest of the customer cannot purchase the remaining rice, as their demand is greater than available amount.

Approach:
In order to meet the demand of maximum number of customers we must start with the customer with minimum demand so that we have maximum amount of rice left to satisfy remaining customers. Therefore, sort the customers according to the increasing order of demand so that maximum number of customers can be satisfied.

Below is the implementation of above approach:

```
 // CPP program to find maximum number
// of customers that can be satisfied
#include <bits/stdc++.h>
using namespace std;

vector<pair<long long, int> > v;
```

```
// print maximum number of satisfied
// customers and their indexes
void solve(int n, int d, int a, int b,
                        int arr[][2])
{
    // Creating an vector of pair of
    // total demand and customer number
    for (int i = 0; i < n; i++) {
        int m = arr[i][0], t = arr[i][1];
        v.push_back(make_pair((a * m + b * t),
                                    i + 1));
    }

    // Sorting the customers according
    // to their total demand
    sort(v.begin(), v.end());

    vector<int> ans;

    // Taking the first k customers that
    // can be satisfied by total amount d
    for (int i = 0; i < n; i++) {
        if (v[i].first <= d) {
            ans.push_back(v[i].second);
            d -= v[i].first;
        }
    }

    cout << ans.size() << endl;
    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << " ";
}

// Driver program
int main()
{
    // Initializing variables
    int n = 5;
    long d = 5;
    int a = 1, b = 1;
    int arr[][2] = {{2, 0},
                    {3, 2},
                    {4, 4},
                    {10, 0},
                    {0, 1}};

    solve(n, d, a, b, arr);
    return 0;
```

```
}
```

Output:

```
2
5 1
```

Source

https://www.geeksforgeeks.org/maximum-number-customers-can-satisfied-given-quantity/

Chapter 70

Maximum product subset of an array

Maximum product subset of an array - GeeksforGeeks

Given an array a, we have to find maximum product possible with the subset of elements present in the array. The maximum product can be single element also.

Examples:

```
Input : a[] = { -1, -1, -2, 4, 3 }
Output : 24
Explanation : Maximum product will be ( -2 * -1 * 4 * 3 ) = 24

Input : a[] = { -1, 0 }
Output : 0
Explanation : 0(single element) is maximum product possible

Input : a[] = { 0, 0, 0 }
Output : 0
```

A **simple solution** is to generate all subsets, find product of every subset and return maximum product.

A **better solution** is to use the below facts.

1. If there are even number of negative numbers and no zeros, result is simply product of all
2. If there are odd number of negative numbers and no zeros, result is product of all except the largest valued negative number.

3. If there are zeros, result is product of all except these zeros with one exceptional case. The exceptional case is when there is one negative number and all other elements are 0. In this case, result is 0.

```
 // CPP program to find maximum product of
// a subset.
#include <bits/stdc++.h>
using namespace std;

int maxProductSubset(int a[], int n)
{
    if (n == 1)
        return a[0];

    // Find count of negative numbers, count
    // of zeros, maximum valued negative number
    // and product of non-zero numbers
    int max_neg = INT_MIN;
    int count_neg = 0, count_zero = 0;
    int prod = 1;
    for (int i = 0; i < n; i++) {

        // If number is 0, we don't
        // multiply it with product.
        if (a[i] == 0) {
            count_zero++;
            continue;
        }

        // Count negatives and keep
        // track of maximum valued negative.
        if (a[i] < 0) {
            count_neg++;
            max_neg = max(max_neg, a[i]);
        }

        prod = prod * a[i];
    }

    // If there are all zeros
    if (count_zero == n)
        return 0;

    // If there are odd number of
    // negative numbers
    if (count_neg & 1) {

        // Exceptional case: There is only
```

```
        // negative and all other are zeros
        if (count_neg == 1 &&
            count_zero > 0 &&
            count_zero + count_neg == n)
            return 0;

        // Otherwise result is product of
        // all non-zeros divided by maximum
        // valued negative.
        prod = prod / max_neg;
    }

    return prod;
}

int main()
{
    int a[] = {  -1, -1, -2, 4, 3  };
    int n = sizeof(a) / sizeof(a[0]);
    cout << maxProductSubset(a, n);
    return 0;
}
```

Output:

```
24
```

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

Source

https://www.geeksforgeeks.org/maximum-product-subset-array/

Chapter 71

Maximum sum of absolute difference of an array

Maximum sum of absolute difference of an array - GeeksforGeeks

Given an array, we need to find the maximum sum of absolute difference of the array elements for a sequence of this array.

Examples:

```
Input : { 1, 2, 4, 8 }
Output : 18
Explanation : For the given array there are
several sequence possible
like : {2, 1, 4, 8}
       {4, 2, 1, 8} and some more.
Now, the absolute difference of an array sequence will be
like for this array sequence {1, 2, 4, 8}, the absolute
difference sum is
= |1-2| + |2-4| + |4-8| + |8-1|
= 14
For the given array, we get the maximum value for
the sequence {1, 8, 2, 4}
= |1-8| + |8-2| + |2-4| + |4-1|
= 18
```

To solve this problem, we have to think greedily that how can we maximize the difference value of the elements so that we can have a maximum sum. This is possible only if we calculate the difference between some very high values and some very low values like (highest – smallest). This is the idea which we have to use to solve this problem. Let us see the above example, we will have maximum difference possible for sequence {1, 8, 2, 4} because in this sequence we will get some high difference values, (1-8 = 7, 8-2 = 6 ..). Here, by

placing 8(highest element) in place of 1 and 2 we get two high difference values. Similarly, for the other values, we will place next highest values in between other, as we have only one left i.e 4 which is placed at last.

Algorithm: *To get the maximum sum, we should have a sequence in which small and large elements comes alternate. This is done to get maximum difference.*

For the implementation of the above algorithm ->
1. We will sort the array.
2. Calculate the final sequence by taking one smallest element and largest element from the sorted array and make one vector array of this final sequence.
3. Finally, calculate the sum of absolute difference between the elements of the array.

Below is the implementation of above idea :
C++

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

int MaxSumDifference(int a[], int n)
{
    // final sequence stored in the vector
    vector<int> finalSequence;

    // sort the original array
    // so that we can retrieve
    // the large elements from
    // the end of array elements
    sort(a, a + n);

    // In this loop first we will insert
    // one smallest element not entered
    // till that time in final sequence
    // and then enter a highest element
    // (not entered till that time) in
    // final sequence so that we
    // have large difference value. This
    // process is repeated till all array
    // has completely entered in sequence.
    // Here, we have loop till n/2 because
    // we are inserting two elements at a
    // time in loop.
    for (int i = 0; i < n / 2; ++i) {
        finalSequence.push_back(a[i]);
        finalSequence.push_back(a[n - i - 1]);
    }
```

```
    // variable to store the
    // maximum sum of absolute
    // difference
    int MaximumSum = 0;

    // In this loop absolute difference
    // of elements for the final sequence
    // is calculated.
    for (int i = 0; i < n - 1; ++i) {
        MaximumSum = MaximumSum + abs(finalSequence[i] -
                                  finalSequence[i + 1]);
    }

    // absolute difference of last element
    // and 1st element
    MaximumSum = MaximumSum + abs(finalSequence[n - 1] -
                                      finalSequence[0]);

    // return the value
    return MaximumSum;
}

// Driver function
int main()
{
    int a[] = { 1, 2, 4, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << MaxSumDifference(a, n) << endl;
}
```

Java

```
 // Java implementation of
// above algorithm
import java.io.*;
import java.util.*;

public class GFG {

    static int MaxSumDifference(Integer []a, int n)
    {

        // final sequence stored in the vector
        List<Integer> finalSequence =
                        new ArrayList<Integer>();

        // sort the original array
```

```
    // so that we can retrieve
    // the large elements from
    // the end of array elements
    Arrays.sort(a);

    // In this loop first we will insert
    // one smallest element not entered
    // till that time in final sequence
    // and then enter a highest element
    // (not entered till that time) in
    // final sequence so that we
    // have large difference value. This
    // process is repeated till all array
    // has completely entered in sequence.
    // Here, we have loop till n/2 because
    // we are inserting two elements at a
    // time in loop.
    for (int i = 0; i < n / 2; ++i) {
        finalSequence.add(a[i]);
        finalSequence.add(a[n - i - 1]);
    }

    // variable to store the
    // maximum sum of absolute
    // difference
    int MaximumSum = 0;

    // In this loop absolute difference
    // of elements for the final sequence
    // is calculated.
    for (int i = 0; i < n - 1; ++i) {
        MaximumSum = MaximumSum +
              Math.abs(finalSequence.get(i)
               - finalSequence.get(i + 1));
    }

    // absolute difference of last element
    // and 1st element
    MaximumSum = MaximumSum +
          Math.abs(finalSequence.get(n - 1)
                   - finalSequence.get(0));

    // return the value
    return MaximumSum;
}

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

```
    {
        Integer []a = { 1, 2, 4, 8 };
        int n = a.length;

        System.out.print(MaxSumDifference(a, n));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
 import numpy as np
class GFG:

    def MaxSumDifference(a,n):
        # sort the original array
        # so that we can retrieve
        # the large elements from
        # the end of array elements
        np.sort(a);

        # In this loop first we will
        # insert one smallest element
        # not entered till that time
        # in final sequence and then
        # enter a highest element(not
        # entered till that time) in
        # final sequence so that we
        # have large difference value.
        # This process is repeated till
        # all array has completely
        # entered in sequence. Here,
        # we have loop till n/2 because
        # we are inserting two elements
        # at a time in loop.
        j = 0
        finalSequence = [0 for x in range(n)]
        for i in range(0, int(n / 2)):
            finalSequence[j] = a[i]
            finalSequence[j + 1] = a[n - i - 1]
            j = j + 2

        # variable to store the
        # maximum sum of absolute
        # difference
        MaximumSum = 0
```

```
        # In this loop absolute
        # difference of elements
        # for the final sequence
        # is calculated.
        for i in range(0, n - 1):
            MaximumSum = (MaximumSum +
                          abs(finalSequence[i] -
                              finalSequence[i + 1]))

        # absolute difference of last
        # element and 1st element
        MaximumSum = (MaximumSum +
                      abs(finalSequence[n - 1] -
                         finalSequence[0]));

        # return the value
        print (MaximumSum)

# Driver Code
a = [ 1, 2, 4, 8 ]
n = len(a)
GFG.MaxSumDifference(a, n);

# This code is contributed
# by Prateek Bajaj
```

C#

```
 // C# implementation of
// above algorithm
using System;
using System.Collections.Generic;
class GFG {

    static int MaxSumDifference(int []a, int n)
    {

        // final sequence stored in the vector
        List<int> finalSequence = new List<int>();

        // sort the original array
        // so that we can retrieve
        // the large elements from
        // the end of array elements
        Array.Sort(a);

        // In this loop first we will insert
```

```
        // one smallest element not entered
        // till that time in final sequence
        // and then enter a highest element
        // (not entered till that time) in
        // final sequence so that we
        // have large difference value. This
        // process is repeated till all array
        // has completely entered in sequence.
        // Here, we have loop till n/2 because
        // we are inserting two elements at a
        // time in loop.
        for (int i = 0; i < n / 2; ++i) {
            finalSequence.Add(a[i]);
            finalSequence.Add(a[n - i - 1]);
        }

        // variable to store the
        // maximum sum of absolute
        // difference
        int MaximumSum = 0;

        // In this loop absolute difference
        // of elements for the final sequence
        // is calculated.
        for (int i = 0; i < n - 1; ++i) {
            MaximumSum = MaximumSum + Math.Abs(finalSequence[i] -
                                    finalSequence[i + 1]);
        }

        // absolute difference of last element
        // and 1st element
        MaximumSum = MaximumSum + Math.Abs(finalSequence[n - 1] -
                                        finalSequence[0]);

        // return the value
        return MaximumSum;
    }

    // Driver Code
    public static void Main()
    {
        int []a = { 1, 2, 4, 8 };
        int n = a.Length;

        Console.WriteLine(MaxSumDifference(a, n));
    }
}
```

```
// This code is contributed by
// Manish Shaw (manishshaw1)
```

Output :

```
18
```

Improved By : manishshaw1, Prateek Bajaj

Source

https://www.geeksforgeeks.org/maximum-sum-absolute-difference-array/

Chapter 72

Maximum sum of increasing order elements from n arrays

Maximum sum of increasing order elements from n arrays - GeeksforGeeks

Given n arrays of size m eacg. Find maximum sum obtained by selecting a number from each array such that the elements selected from i-th array is more than the element selected from (i-1)-th array. If maximum sum can not be obtained then return 0.

Examples:

```
Input : arr[][] = {{1, 7, 3, 4},
                   {4, 2, 5, 1},
                   {9, 5, 1, 8}}
Output : 18
Explanation :
We can select 4 from first array, 5 from
second array and 9 from third array.

Input : arr[][] = {{9, 8, 7},
                   {6, 5, 4},
                   {3, 2, 1}}
Output : 0
```

The idea is to start picking from last array. We pick the maximum element from last array, then we move to second last array. In second last array, we find the largest element which is smaller than the maximum element picked from last array. We repeat this process till we reach first array.

To obtain maximum sum we can sort all arrays and start bottom to up traversing each array from right to left and choose a number such that it is greater then previous element. If we are not able to select an element from array then return 0.

C++

```
 // CPP program to find maximum sum
// by selecting a element from n arrays
#include <bits/stdc++.h>
#define M 4
using namespace std;

// To calculate maximum sum by
// selecting element from each array
int maximumSum(int a[][M], int n) {

  // Sort each array
  for (int i = 0; i < n; i++)
    sort(a[i], a[i] + M);

  // Store maximum element
  // of last array
  int sum = a[n - 1][M - 1];
  int prev = a[n - 1][M - 1];
  int i, j;

  // Selecting maximum element from
  // previoulsy selected element
  for (i = n - 2; i >= 0; i--) {
    for (j = M - 1; j >= 0; j--) {
      if (a[i][j] < prev) {
        prev = a[i][j];
        sum += prev;
        break;
      }
    }

    // j = -1 means no element is
    // found in a[i] so return 0
    if (j == -1)
      return 0;
  }

  return sum;
}

// Driver program to test maximumSum
int main() {
  int arr[][M] = {{1, 7, 3, 4},
                  {4, 2, 5, 1},
                  {9, 5, 1, 8}};
  int n = sizeof(arr) / sizeof(arr[0]);
  cout << maximumSum(arr, n);
  return 0;
```

```
}
```

Java

```
 // Java program to find
// maximum sum by selecting
// a element from n arrays
import java.io.*;

class GFG
{
    static int M = 4;
    static int arr[][] = {{1, 7, 3, 4},
                          {4, 2, 5, 1},
                          {9, 5, 1, 8}};

    static void sort(int a[][],
                     int row, int n)
    {
        for (int i = 0; i < M - 1; i++)
        {
            if(a[row][i] > a[row][i + 1])
            {
                int temp = a[row][i];
                a[row][i] = a[row][i + 1];
                a[row][i + 1] = temp;
            }
        }
    }

    // To calculate maximum
    // sum by selecting element
    // from each array
    static int maximumSum(int a[][],
                          int n)
    {

    // Sort each array
    for (int i = 0; i < n; i++)
        sort(a, i, n);

    // Store maximum element
    // of last array
    int sum = a[n - 1][M - 1];
    int prev = a[n - 1][M - 1];
    int i, j;

    // Selecting maximum element
```

```
    // from previoulsy selected
    // element
    for (i = n - 2; i >= 0; i--)
    {
        for (j = M - 1; j >= 0; j--)
        {
            if (a[i][j] < prev)
            {
                prev = a[i][j];
                sum += prev;
                break;
            }
        }

        // j = -1 means no element
        // is found in a[i] so
        // return 0
        if (j == -1)
        return 0;
    }
    return sum;
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = arr.length;
        System.out.print(maximumSum(arr, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python3 program to find
# maximum sum by selecting
# a element from n arrays
M = 4;

# To calculate maximum sum
# by selecting element from
# each array
def maximumSum(a, n) :

    global M;
```

```
    # Sort each array
    for i in range(0, n) :
        a[i].sort();

    # Store maximum element
    # of last array
    sum = a[n - 1][M - 1];
    prev = a[n - 1][M - 1];

    # Selecting maximum
    # element from previoulsy
    # selected element
    for i in range(n - 2,
                   -1, -1) :

        for j in range(M - 1,
                       -1, -1) :

            if (a[i][j] < prev) :

                prev = a[i][j];
                sum += prev;
                break;

        # j = -1 means no element
        # is found in a[i] so
        # return 0
        if (j == -1) :
            return 0;
    return sum;

# Driver Code
arr = [[1, 7, 3, 4],
       [4, 2, 5, 1],
       [9, 5, 1, 8]];
n = len(arr) ;
print (maximumSum(arr, n));

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find maximum
// sum by selecting a element
// from n arrays
using System;
```

```
class GFG
{
    static int M = 4;

    static void sort(ref int[,] a,
                     int row, int n)
    {
        for (int i = 0; i < M-1; i++)
        {
            if(a[row, i] > a[row, i + 1])
            {
                int temp = a[row, i];
                a[row, i] = a[row, i + 1];
                a[row, i + 1] = temp;
            }
        }
    }

    // To calculate maximum
    // sum by selecting
    // element from each array
    static int maximumSum(int[,] a,
                          int n)
    {

    int i = 0, j = 0;

    // Sort each array
    for (i = 0; i < n; i++)
        sort(ref a, i, n);

    // Store maximum element
    // of last array
    int sum = a[n - 1, M - 1];
    int prev = a[n - 1, M - 1];


    // Selecting maximum element
    // from previoulsy selected
    // element
    for (i = n - 2; i >= 0; i--)
    {
        for (j = M - 1; j >= 0; j--)
        {
        if (a[i, j] < prev)
        {
            prev = a[i, j];
            sum += prev;
```

```
            break;
        }
        }

        // j = -1 means no element
        // is found in a[i] so
        // return 0
        if (j == -1)
        return 0;
    }

    return sum;
    }

    // Driver Code
    static void Main()
    {
    int [,]arr = new int[,]{{1, 7, 3, 4},
                            {4, 2, 5, 1},
                            {9, 5, 1, 8}};
    int n = arr.GetLength(0);
    Console.Write(maximumSum(arr, n));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// PHP program to find maximum
// sum by selecting a element
// from n arrays
$M = 4;

// To calculate maximum sum
// by selecting element from
// each array
function maximumSum($a, $n)
{
    global $M;

    // Sort each array
    for ($i = 0; $i < $n; $i++)
        sort($a[$i]);

    // Store maximum element
```

```
    // of last array
    $sum = $a[$n - 1][$M - 1];
    $prev = $a[$n - 1][$M - 1];
    $i; $j;

    // Selecting maximum element from
    // previoulsy selected element
    for ($i = $n - 2; $i >= 0; $i--)
    {
        for ($j = $M - 1; $j >= 0; $j--)
        {
        if ($a[$i][$j] < $prev)
        {
            $prev = $a[$i][$j];
            $sum += $prev;
            break;
        }
        }

        // j = -1 means no element is
        // found in a[i] so return 0
        if ($j == -1)
        return 0;
    }

    return $sum;
}

// Driver Code
$arr = array(array(1, 7, 3, 4),
             array(4, 2, 5, 1),
             array(9, 5, 1, 8));
$n = sizeof($arr) ;
echo maximumSum($arr, $n);

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

Output:

```
18
```

Worst Case Time Complexity : O(mn Log m)

We can optimize above solution to work in O(mn). We can skip sorting to find the maximum elements.

C++

```
 // CPP program to find maximum sum
// by selecting a element from n arrays
#include <bits/stdc++.h>
#define M 4
using namespace std;

// To calculate maximum sum by
// selecting element from each array
int maximumSum(int a[][M], int n) {

  // Store maximum element of last array
  int prev = *max_element(&a[n-1][0],
                  &a[n-1][M-1] + 1);

  // Selecting maximum element from
  // previoulsy selected element
  int sum = prev;
  for (int i = n - 2; i >= 0; i--) {

    int max_smaller = INT_MIN;
    for (int j = M - 1; j >= 0; j--) {
      if (a[i][j] < prev &&
          a[i][j] > max_smaller)
        max_smaller = a[i][j];
    }

    // max_smaller equals to INT_MIN means
    // no element is found in a[i] so
    // return 0
    if (max_smaller == INT_MIN)
      return 0;

    prev = max_smaller;
    sum += max_smaller;
  }

  return sum;
}

// Driver program to test maximumSum
int main() {
  int arr[][M] = {{1, 7, 3, 4},
                  {4, 2, 5, 1},
                  {9, 5, 1, 8}};
  int n = sizeof(arr) / sizeof(arr[0]);
  cout << maximumSum(arr, n);
  return 0;
}
```

Output:

```
18
```

Time Complexity: O(mn)

Improved By : jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/maximum-sum-increasing-order-elements-n-arrays/

Chapter 73

Maximum trains for which stoppage can be provided

Maximum trains for which stoppage can be provided - GeeksforGeeks

We are given n-platform and two main running railway track for both direction. Trains which needs to stop at your station must occupy one platform for their stoppage and the trains which need not to stop at your station will run away through either of main track without stopping. Now, each train has three value first arrival time, second departure time and third required platform number. We are given m such trains you have to tell maximum number of train for which you can provide stoppage at your station.

Examples:

```
Input : n = 3, m = 6
Train no.|  Arrival Time |Dept. Time | Platform No.
    1    |   10:00       |  10:30    |    1
    2    |   10:10       |  10:30    |    1
    3    |   10:00       |  10:20    |    2
    4    |   10:30       |  12:30    |    2
    5    |   12:00       |  12:30    |    3
    6    |   09:00       |  10:05    |    1
Output : Maximum Stopped Trains = 5
Explanation : If train no. 1 will left
to go without stoppage then 2 and 6 can
easily be accommodated on platform 1.
And 3 and 4 on platform 2 and 5 on platform 3.

Input : n = 1, m = 3
Train no.|Arrival Time|Dept. Time | Platform No.
    1    | 10:00      |  10:30    |    1
    2    | 11:10      |  11:30    |    1
```

```
    3    | 12:00      |  12:20    |    1

Output : Maximum Stopped Trains = 3
Explanation : All three trains can be easily
stopped at platform 1.
```

If we start with a single platform only then we have 1 platform and some trains with their arrival time and departure time and we have to maximize the number of trains on that platform. This task is similar as Activity Selection Problem. So, for n platforms we will simply make n-vectors and put the respective trains in those vectors according to platform number. After that by applying greedy approach we easily solve this problem.
Note : We will take input in form of 4-digit integer for arrival and departure time as 1030 will represent 10:30 so that we may handle the data type easily.
Also, we will choose a 2-D array for input as arr[m][3] where arr[i][0] denotes arrival time, arr[i][1] denotes departure time and arr[i][2] denotes the platform for ith train.

```
 // CPP to design platform for maximum stoppage
#include <bits/stdc++.h>
using namespace std;

// number of platforms and trains
#define n 2
#define m 5

// function to calculate maximum trains stoppage
int maxStop(int arr[][3])
{
    // declaring vector of pairs for platform
    vector<pair<int, int> > vect[n + 1];

    // Entering values in vector of pairs
    // as per platform number
    // make departure time first element
    // of pair
    for (int i = 0; i < m; i++)
        vect[arr[i][2]].push_back(
             make_pair(arr[i][1], arr[i][0]));

    // sort trains for each platform as per
    // dept. time
    for (int i = 0; i <= n; i++)
        sort(vect[i].begin(), vect[i].end());

    // perform activity selection approach
    int count = 0;
    for (int i = 0; i <= n; i++) {
        if (vect[i].size() == 0)
            continue;
```

```
        // first train for each platform will
        // also be selected
        int x = 0;
        count++;
        for (int j = 1; j < vect[i].size(); j++) {
            if (vect[i][j].second >=
                            vect[i][x].first) {
                x = j;
                count++;
            }
        }
    }
    return count;
}

// driver function
int main()
{
    int arr[m][3] = { 1000, 1030, 1,
                      1010, 1020, 1,
                      1025, 1040, 1,
                      1130, 1145, 2,
                      1130, 1140, 2 };
    cout << "Maximum Stopped Trains = "
         << maxStop(arr);
    return 0;
}
```

Output:

```
Maximum Stopped Trains = 3
```

Source

https://www.geeksforgeeks.org/maximum-trains-stoppage-can-provided/

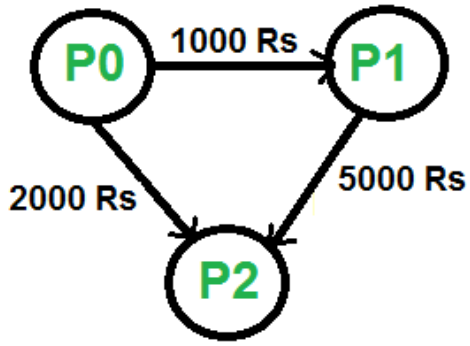
Chapter 74

Minimize Cash Flow among a given set of friends who have borrowed money from each other

Minimize Cash Flow among a given set of friends who have borrowed money from each other - GeeksforGeeks

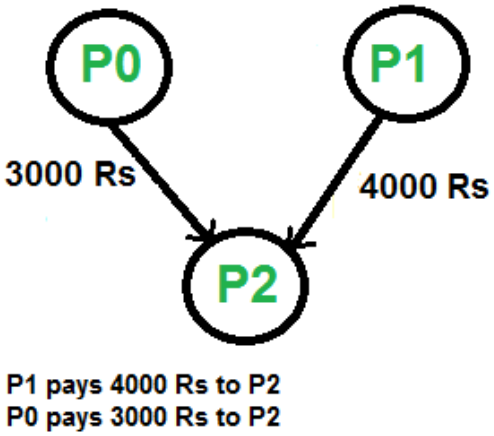
Given a number of friends who have to give or take some amount of money from one another. Design an algorithm by which the total cash flow among all the friends is minimized.

Example:
Following diagram shows input debts to be settled.



P0 has to pay 1000 Rs to P1
P0 also has to pay 2000 Rs to P2
P1 has to pay 5000 Rs to P2.

Above debts can be settled in following optimized way



The idea is to use Greedy algorithm where at every step, settle all amounts of one person and recur for remaining n-1 persons.
How to pick the first person? To pick the first person, calculate the net amount for every person where net amount is obtained by subtracting all debts (amounts to pay) from all credits (amounts to be paid). Once net amount for every person is evaluated, find two persons with maximum and minimum net amounts. These two persons are the most creditors and debtors. The person with minimum of two is our first person to be settled and removed from list. Let the minimum of two amounts be x. We pay 'x' amount from the maximum debtor to maximum creditor and settle one person. If x is equal to the maximum debit, then maximum debtor is settled, else maximum creditor is settled.

The following is detailed algorithm.

Do following for every person Pi where i is from 0 to n-1.
1) Compute the net amount for every person. The net amount for person 'i' can be computed be subtracting sum of all debts from sum of all credits.

2) Find the two persons that are maximum creditor and maximum debtor. Let the maximum amount to be credited maximum creditor be maxCredit and maximum amount to be debited from maximum debtor be maxDebit. Let the maximum debtor be Pd and maximum creditor be Pc.

3) Find the minimum of maxDebit and maxCredit. Let minimum of two be x. Debit 'x' from Pd and credit this amount to Pc

4) If x is equal to maxCredit, then remove Pc from set of persons and recur for remaining (n-1) persons.

5) If x is equal to maxDebit, then remove Pd from set of persons and recur for remaining (n-1) persons.

Thanks to Balaji S for suggesting this method in a comment here.

The following is implementation of above algorithm.
C++

```
// C++ program to fin maximum cash flow among a set of persons
```

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

// Number of persons (or vertices in the graph)
#define N 3

// A utility function that returns index of minimum value in arr[]
int getMin(int arr[])
{
    int minInd = 0;
    for (int i=1; i<N; i++)
        if (arr[i] < arr[minInd])
            minInd = i;
    return minInd;
}

// A utility function that returns index of maximum value in arr[]
int getMax(int arr[])
{
    int maxInd = 0;
    for (int i=1; i<N; i++)
        if (arr[i] > arr[maxInd])
            maxInd = i;
    return maxInd;
}

// A utility function to return minimum of 2 values
int minOf2(int x, int y)
{
    return (x<y)? x: y;
}

// amount[p] indicates the net amount to be credited/debited
// to/from person 'p'
// If amount[p] is positive, then i'th person will amount[i]
// If amount[p] is negative, then i'th person will give  -amount[i]
void minCashFlowRec(int amount[])
{
    // Find the indexes of minimum and maximum values in amount[]
    // amount[mxCredit] indicates the maximum amount to be given
    //                  (or credited) to any person .
    // And amount[mxDebit] indicates the maximum amount to be taken
    //                  (or debited) from any person.
    // So if there is a positive value in amount[], then there must
    // be a negative value
    int mxCredit = getMax(amount), mxDebit = getMin(amount);

    // If both amounts are 0, then all amounts are settled
```

```
    if (amount[mxCredit] == 0 && amount[mxDebit] == 0)
        return;

    // Find the minimum of two amounts
    int min = minOf2(-amount[mxDebit], amount[mxCredit]);
    amount[mxCredit] -= min;
    amount[mxDebit] += min;

    // If minimum is the maximum amount to be
    cout << "Person " << mxDebit << " pays " << min
         << " to " << "Person " << mxCredit << endl;

    // Recur for the amount array.  Note that it is guaranteed that
    // the recursion would terminate as either amount[mxCredit]
    // or  amount[mxDebit] becomes 0
    minCashFlowRec(amount);
}

// Given a set of persons as graph[] where graph[i][j] indicates
// the amount that person i needs to pay person j, this function
// finds and prints the minimum cash flow to settle all debts.
void minCashFlow(int graph[][N])
{
    // Create an array amount[], initialize all value in it as 0.
    int amount[N] = {0};

    // Calculate the net amount to be paid to person 'p', and
    // stores it in amount[p]. The value of amount[p] can be
    // calculated by subtracting debts of 'p' from credits of 'p'
    for (int p=0; p<N; p++)
       for (int i=0; i<N; i++)
          amount[p] += (graph[i][p] -  graph[p][i]);

    minCashFlowRec(amount);
}

// Driver program to test above function
int main()
{
    // graph[i][j] indicates the amount that person i needs to
    // pay person j
    int graph[N][N] = { {0, 1000, 2000},
                        {0, 0, 5000},
                        {0, 0, 0},};

    // Print the solution
    minCashFlow(graph);
    return 0;
```

```
}
```

Java

```
 // Java program to fin maximum cash
// flow among a set of persons

class GFG
{
    // Number of persons (or vertices in the graph)
    static final int N = 3;

    // A utility function that returns
    // index of minimum value in arr[]
    static int getMin(int arr[])
    {
        int minInd = 0;
        for (int i = 1; i < N; i++)
            if (arr[i] < arr[minInd])
                minInd = i;
        return minInd;
    }

    // A utility function that returns
    // index of maximum value in arr[]
    static int getMax(int arr[])
    {
        int maxInd = 0;
        for (int i = 1; i < N; i++)
            if (arr[i] > arr[maxInd])
                maxInd = i;
        return maxInd;
    }

    // A utility function to return minimum of 2 values
    static int minOf2(int x, int y)
    {
        return (x < y) ? x: y;
    }

    // amount[p] indicates the net amount
    // to be credited/debited to/from person 'p'
    // If amount[p] is positive, then
    // i'th person will amount[i]
    // If amount[p] is negative, then
    // i'th person will give -amount[i]
    static void minCashFlowRec(int amount[])
    {
```

```
    // Find the indexes of minimum and
    // maximum values in amount[]
    // amount[mxCredit] indicates the maximum amount
    // to be given (or credited) to any person .
    // And amount[mxDebit] indicates the maximum amount
    // to be taken(or debited) from any person.
    // So if there is a positive value in amount[],
    // then there must be a negative value
    int mxCredit = getMax(amount), mxDebit = getMin(amount);

    // If both amounts are 0, then
    // all amounts are settled
    if (amount[mxCredit] == 0 && amount[mxDebit] == 0)
        return;

    // Find the minimum of two amounts
    int min = minOf2(-amount[mxDebit], amount[mxCredit]);
    amount[mxCredit] -= min;
    amount[mxDebit] += min;

    // If minimum is the maximum amount to be
    System.out.println("Person " + mxDebit + " pays " + min
                           + " to " + "Person " + mxCredit);

    // Recur for the amount array.
    // Note that it is guaranteed that
    // the recursion would terminate
    // as either amount[mxCredit]  or
    // amount[mxDebit] becomes 0
    minCashFlowRec(amount);
}

// Given a set of persons as graph[]
// where graph[i][j] indicates
// the amount that person i needs to
// pay person j, this function
// finds and prints the minimum
// cash flow to settle all debts.
static void minCashFlow(int graph[][])
{
    // Create an array amount[],
    // initialize all value in it as 0.
    int amount[]=new int[N];

    // Calculate the net amount to
    // be paid to person 'p', and
    // stores it in amount[p]. The
    // value of amount[p] can be
```

```
        // calculated by subtracting
        // debts of 'p' from credits of 'p'
        for (int p = 0; p < N; p++)
        for (int i = 0; i < N; i++)
            amount[p] += (graph[i][p] - graph[p][i]);

        minCashFlowRec(amount);
    }

    // Driver code
    public static void main (String[] args)
    {
        // graph[i][j] indicates the amount
        // that person i needs to pay person j
        int graph[][] = { {0, 1000, 2000},
                          {0, 0, 5000},
                          {0, 0, 0},};

        // Print the solution
        minCashFlow(graph);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to fin maximum
# cash flow among a set of persons

# Number of persons(or vertices in graph)
N = 3

# A utility function that returns
# index of minimum value in arr[]
def getMin(arr):

    minInd = 0
    for i in range(1, N):
        if (arr[i] < arr[minInd]):
            minInd = i
    return minInd

# A utility function that returns
# index of maximum value in arr[]
def getMax(arr):

    maxInd = 0
```

```
    for i in range(1, N):
        if (arr[i] > arr[maxInd]):
            maxInd = i
    return maxInd

# A utility function to
# return minimum of 2 values
def minOf2(x, y):

    return x if x < y else y

# amount[p] indicates the net amount to
# be credited/debited to/from person 'p'
# If amount[p] is positive, then i'th
# person will amount[i]
# If amount[p] is negative, then i'th
# person will give -amount[i]
def minCashFlowRec(amount):

    # Find the indexes of minimum
    # and maximum values in amount[]
    # amount[mxCredit] indicates the maximum
    # amount to be given(or credited) to any person.
    # And amount[mxDebit] indicates the maximum amount
    # to be taken (or debited) from any person.
    # So if there is a positive value in amount[],
    # then there must be a negative value
    mxCredit = getMax(amount)
    mxDebit = getMin(amount)

    # If both amounts are 0,
    # then all amounts are settled
    if (amount[mxCredit] == 0 and amount[mxDebit] == 0):
        return 0

    # Find the minimum of two amounts
    min = minOf2(-amount[mxDebit], amount[mxCredit])
    amount[mxCredit] -=min
    amount[mxDebit] += min

    # If minimum is the maximum amount to be
    print("Person " , mxDebit , " pays " , min
        , " to " , "Person " , mxCredit)

    # Recur for the amount array. Note that
    # it is guaranteed that the recursion
    # would terminate as either amount[mxCredit]
    # or amount[mxDebit] becomes 0
```

```
    minCashFlowRec(amount)

# Given a set of persons as graph[] where
# graph[i][j] indicates the amount that
# person i needs to pay person j, this
# function finds and prints the minimum
# cash flow to settle all debts.
def minCashFlow(graph):

    # Create an array amount[],
    # initialize all value in it as 0.
    amount = [0 for i in range(N)]

    # Calculate the net amount to be paid
    # to person 'p', and stores it in amount[p].
    # The value of amount[p] can be calculated by
    # subtracting debts of 'p' from credits of 'p'
    for p in range(N):
        for i in range(N):
            amount[p] += (graph[i][p] - graph[p][i])

    minCashFlowRec(amount)

# Driver code

# graph[i][j] indicates the amount
# that person i needs to pay person j
graph = [ [0, 1000, 2000],
          [0, 0, 5000],
          [0, 0, 0] ]

minCashFlow(graph)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to fin maximum cash
// flow among a set of persons
using System;

class GFG
{
    // Number of persons (or
    // vertices in the graph)
    static int N = 3;

    // A utility function that returns
```

```
// index of minimum value in arr[]
static int getMin(int []arr)
{
    int minInd = 0;
    for (int i = 1; i < N; i++)
        if (arr[i] < arr[minInd])
            minInd = i;
    return minInd;
}

// A utility function that returns
// index of maximum value in arr[]
static int getMax(int []arr)
{
    int maxInd = 0;
    for (int i = 1; i < N; i++)
        if (arr[i] > arr[maxInd])
            maxInd = i;
    return maxInd;
}

// A utility function to return
// minimum of 2 values
static int minOf2(int x, int y)
{
    return (x < y) ? x: y;
}

// amount[p] indicates the net amount
// to be credited/debited to/from person 'p'
// If amount[p] is positive, then
// i'th person will amount[i]
// If amount[p] is negative, then
// i'th person will give -amount[i]
static void minCashFlowRec(int []amount)
{
    // Find the indexes of minimum and
    // maximum values in amount[]
    // amount[mxCredit] indicates the maximum amount
    // to be given (or credited) to any person .
    // And amount[mxDebit] indicates the maximum amount
    // to be taken(or debited) from any person.
    // So if there is a positive value in amount[],
    // then there must be a negative value
    int mxCredit = getMax(amount), mxDebit = getMin(amount);

    // If both amounts are 0, then
    // all amounts are settled
```

```
    if (amount[mxCredit] == 0 &&
        amount[mxDebit] == 0)
        return;

    // Find the minimum of two amounts
    int min = minOf2(-amount[mxDebit], amount[mxCredit]);
    amount[mxCredit] -= min;
    amount[mxDebit] += min;

    // If minimum is the maximum amount to be
    Console.WriteLine("Person " + mxDebit +
                      " pays " + min + " to " +
                      "Person " + mxCredit);

    // Recur for the amount array.
    // Note that it is guaranteed that
    // the recursion would terminate
    // as either amount[mxCredit] or
    // amount[mxDebit] becomes 0
    minCashFlowRec(amount);
}

// Given a set of persons as graph[]
// where graph[i][j] indicates
// the amount that person i needs to
// pay person j, this function
// finds and prints the minimum
// cash flow to settle all debts.
static void minCashFlow(int [,]graph)
{
    // Create an array amount[],
    // initialize all value in it as 0.
    int []amount=new int[N];

    // Calculate the net amount to
    // be paid to person 'p', and
    // stores it in amount[p]. The
    // value of amount[p] can be
    // calculated by subtracting
    // debts of 'p' from credits of 'p'
    for (int p = 0; p < N; p++)
    for (int i = 0; i < N; i++)
        amount[p] += (graph[i,p] - graph[p,i]);

    minCashFlowRec(amount);
}

// Driver code
```

```
    public static void Main ()
    {
        // graph[i][j] indicates the amount
        // that person i needs to pay person j
        int [,]graph = { {0, 1000, 2000},
                         {0, 0, 5000},
                         {0, 0, 0},};

        // Print the solution
        minCashFlow(graph);
    }
}

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

Output:

```
Person 1 pays 4000 to Person 2
Person 0 pays 3000 to Person 2
```

Algorithmic Paradigm: Greedy
Time Complexity: O(N^2) where N is the number of persons.

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

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/minimize-cash-flow-among-given-set-friends-borrowed-money/

Chapter 75

Minimize the maximum difference between the heights

Minimize the maximum difference between the heights - GeeksforGeeks

Given heights of n towers and a value k. We need to either increase or decrease height of every tower by k (only once) where k > 0. The task is to minimize the difference between the heights of the longest and the shortest tower after modifications, and output this difference.

Examples:

```
Input  : arr[] = {1, 15, 10}, k = 6
Output :  Maximum difference is 5.
Explanation : We change 1 to 6, 15 to
9 and 10 to 4. Maximum difference is 5
(between 4 and 9). We can't get a lower
difference.

Input : arr[] = {1, 5, 15, 10}
        k = 3
Output : Maximum difference is 8
arr[] = {4, 8, 12, 7}

Input : arr[] = {4, 6}
        k = 10
Output : Maximum difference is 2
arr[] = {14, 16} OR {-6, -4}

Input : arr[] = {6, 10}
        k = 3
Output : Maximum difference is 2
arr[] = {9, 7}
```

```
Input : arr[] = {1, 10, 14, 14, 14, 15}
        k = 6
Output: Maximum difference is 5
arr[] = {7, 4, 8, 8, 8, 9}

Input : arr[] = {1, 2, 3}
        k = 2
Output: Maximum difference is 2
arr[] = {3, 4, 5}
```

Source : Adobe Interview Experience Set 24 (On-Campus for MTS)

The idea is to sort all elements increasing order.

Below is implementation of above idea.

C++

```
 // C++ program to find the minimum possible
// difference between maximum and minimum
// elements when we have to add/subtract
// every number by k
#include <bits/stdc++.h>
using namespace std;

// Modifies the array by subtracting/adding
// k to every element such that the difference
// between maximum and minimum is minimized
int getMinDiff(int arr[], int n, int k)
{
    if (n == 1)
       return 0;

    // Sort all elements
    sort(arr, arr+n);

    // Initialize result
    int ans = arr[n-1] - arr[0];

    // Handle corner elements
    int small = arr[0] + k;
    int big = arr[n-1] - k;
    if (small > big)
       swap(small, big);

    // Traverse middle elements
    for (int i = 1; i < n-1; i ++)
    {
        int subtract = arr[i] - k;
```

```
        int add = arr[i] + k;

        // If both subtraction and addition
        // do not change diff
        if (subtract >= small || add <= big)
            continue;

        // Either subtraction causes a smaller
        // number or addition causes a greater
        // number. Update small or big using
        // greedy approach (If big - subtract
        // causes smaller diff, update small
        // Else update big)
        if (big - subtract <= add - small)
            small = subtract;
        else
            big = add;
    }

    return  min(ans, big - small);
}

// Driver function to test the above function
int main()
{
    int arr[] = {4, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 10;
    cout << "\nMaximum difference is "
        << getMinDiff(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to find the minimum possible
// difference between maximum and minimum
// elements when we have to add/subtract
// every number by k
import java.util.*;

class GFG {

    // Modifies the array by subtracting/adding
    // k to every element such that the difference
    // between maximum and minimum is minimized
    static int getMinDiff(int arr[], int n, int k)
    {
```

```
    if (n == 1)
    return 0;

    // Sort all elements
    Arrays.sort(arr);

    // Initialize result
    int ans = arr[n-1] - arr[0];

    // Handle corner elements
    int small = arr[0] + k;
    int big = arr[n-1] - k;
    int temp = 0;

    if (small > big)
    {
        temp = small;
        small = big;
        big = temp;
    }

    // Traverse middle elements
    for (int i = 1; i < n-1; i ++)
    {
        int subtract = arr[i] - k;
        int add = arr[i] + k;

        // If both subtraction and addition
        // do not change diff
        if (subtract >= small || add <= big)
            continue;

        // Either subtraction causes a smaller
        // number or addition causes a greater
        // number. Update small or big using
        // greedy approach (If big - subtract
        // causes smaller diff, update small
        // Else update big)
        if (big - subtract <= add - small)
            small = subtract;
        else
            big = add;
    }

    return Math.min(ans, big - small);
}

// Driver function to test the above function
```

```
    public static void main(String[] args)
    {
        int arr[] = {4, 6};
        int n = arr.length;
        int k = 10;
        System.out.println("Maximum difference is "+
                           getMinDiff(arr, n, k));
    }
}
// This code is contributed by Prerna Saini
```

Python3

```
 # Python 3 program to find the minimum
# possible difference between maximum
# and minimum elements when we have to
# add / subtract every number by k

# Modifies the array by subtracting /
# adding k to every element such that
# the difference between maximum and
# minimum is minimized
def getMinDiff(arr, n, k):

    if (n == 1):
        return 0

    # Sort all elements
    arr.sort()

    # Initialize result
    ans = arr[n-1] - arr[0]

    # Handle corner elements
    small = arr[0] + k
    big = arr[n-1] - k

    if (small > big):
        small, big = big, small

    # Traverse middle elements
    for i in range(1, n-1):

        subtract = arr[i] - k
        add = arr[i] + k

        # If both subtraction and addition
        # do not change diff
```

```
        if (subtract >= small or add <= big):
            continue

        # Either subtraction causes a smaller
        # number or addition causes a greater
        # number. Update small or big using
        # greedy approach (If big - subtract
        # causes smaller diff, update small
        # Else update big)
        if (big - subtract <= add - small):
            small = subtract
        else:
            big = add


    return min(ans, big - small)


# Driver function
arr = [ 4, 6 ]
n = len(arr)
k = 10

print("Maximum difference is", getMinDiff(arr, n, k))

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

Output :

```
Maximum difference is 2
```

Time Complexity : O(n Log n)

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

Source

https://www.geeksforgeeks.org/minimize-the-maximum-difference-between-the-heights/

Chapter 76

Minimize the sum of product of two arrays with permutations allowed

Minimize the sum of product of two arrays with permutations allowed - GeeksforGeeks

Given two arrays, A and B, of equal size n, the task is to find the minimum value of A[0] * B[0] + A[1] * B[1] +…+ A[n-1] * B[n-1]. Shuffling of elements of arrays A and B is allowed.

Examples :

```
Input : A[] = {3, 1, 1} and B[] = {6, 5, 4}.
Output : 23
Minimum value of S = 1*6 + 1*5 + 3*4 = 23.

Input : A[] = { 6, 1, 9, 5, 4 } and B[] = { 3, 4, 8, 2, 4 }
Output : 80.
Minimum value of S = 1*8 + 4*4 + 5*4 + 6*3 + 9*2 = 80.
```

The idea is to multiply minimum element of one array to maximum element of another array. Algorithm to solve this problem:

1. Sort both the arrays A and B.
2. Traverse the array and for each element, multiply A[i] and B[n – i – 1] and add to the total.

Below is the implementation of this approach:
C/C++

```
 // C++ program to calculate minimum sum of product
// of two arrays.
#include <bits/stdc++.h>
using namespace std;

// Returns minimum sum of product of two arrays
// with permutations allowed
int minValue(int A[], int B[], int n)
{
    // Sort A and B so that minimum and maximum
    // value can easily be fetched.
    sort(A, A + n);
    sort(B, B + n);

    // Multiplying minimum value of A and maximum
    // value of B
    int result = 0;
    for (int i = 0; i < n; i++)
        result += (A[i] * B[n - i - 1]);

    return result;
}

// Driven Program
int main()
{
    int A[] = { 3, 1, 1 };
    int B[] = { 6, 5, 4 };
    int n = sizeof(A) / sizeof(A[0]);
    cout << minValue(A, B, n) << endl;
    return 0;
}
```

Java

```
 // java program to calculate minimum
// sum of product of two arrays.
import java.io.*;
import java.util.*;

class GFG {

    // Returns minimum sum of product of two arrays
    // with permutations allowed
    static int minValue(int A[], int B[], int n)
    {
        // Sort A and B so that minimum and maximum
        // value can easily be fetched.
```

```
        Arrays.sort(A);
        Arrays.sort(B);

        // Multiplying minimum value of A
        // and maximum value of B
        int result = 0;
        for (int i = 0; i < n; i++)
            result += (A[i] * B[n - i - 1]);

        return result;
    }

    // Driven Program
    public static void main(String[] args)
    {
        int A[] = { 3, 1, 1 };
        int B[] = { 6, 5, 4 };
        int n = A.length;
        ;
        System.out.println(minValue(A, B, n));
    }
}

// This code is contributed by vt_m
```

Python

```
 # Python program to calculate minimum sum of product
# of two arrays.

# Returns minimum sum of product of two arrays
# with permutations allowed
def minValue(A, B, n):
    # Sort A and B so that minimum and maximum
    # value can easily be fetched.
    sorted(A)
    sorted(B)

    # Multiplying minimum value of A and maximum
    # value of B
    result = 0
    for i in range(n):
        result += (A[i] * B[n - i - 1])

    return result

# Driven Program
A = [3, 1, 1]
```

```
B = [6, 5, 4]
n = len(A)
print minValue(A, B, n)

# Contributed by: Afzal Ansari
```

C#

```
 // C# program to calculate minimum
// sum of product of two arrays.
using System;

class GFG {

    // Returns minimum sum of product
    // of two arrays with permutations
    // allowed
    static int minValue(int[] a, int[] b,
                                  int n)
    {

        // Sort A and B so that minimum
        // and maximum value can easily
        // be fetched.
        Array.Sort(a);
        Array.Sort(b);

        // Multiplying minimum value of
        // A and maximum value of B
        int result = 0;

        for (int i = 0; i < n; i++)
            result += (a[i] * b[n - i - 1]);

        return result;
    }

    // Driven Program
    public static void Main()
    {

        int[] a = { 3, 1, 1 };
        int[] b = { 6, 5, 4 };
        int n = a.Length;

        Console.Write(minValue(a, b, n));
    }
}
```

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

PHP

```
 <?php
// PHP program to calculate minimum
// sum of product of two arrays.

// Returns minimum sum of
// product of two arrays
// with permutations allowed
function minValue($A, $B, $n)
{
    // Sort A and B so that minimum
    // and maximum value can easily
    // be fetched.
    sort($A); sort($A , $n);
    sort($B); sort($B , $n);

    // Multiplying minimum value of
    // A and maximum value of B
    $result = 0;
    for ($i = 0; $i < $n; $i++)
        $result += ($A[$i] *
                    $B[$n - $i - 1]);

    return $result;
}

// Driver Code
$A = array( 3, 1, 1 );
$B = array( 6, 5, 4 );
$n = sizeof($A) / sizeof($A[0]);
echo minValue($A, $B, $n) ;

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

Output :

```
23
```

Time Complexity : O(n log n).

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/minimize-sum-product-two-arrays-permutations-allowed/

Chapter 77

Minimum Cost Path with Left, Right, Bottom and Up moves allowed

Minimum Cost Path with Left, Right, Bottom and Up moves allowed - GeeksforGeeks

Given a two dimensional grid, each cell of which contains integer cost which represents a cost to traverse through that cell, we need to find a path from top left cell to bottom right cell by which total cost incurred is minimum.

Note : It is assumed that negative cost cycles do not exist in input matrix.

This problem is extension of below problem.

Min Cost Path with right and bottom moves allowed.

In previous problem only going right and bottom was allowed but in this problem we are allowed to go bottom, up, right and left i.e. in all 4 direction.

Examples:

```
A cost grid is given in below diagram, minimum
cost to reach bottom right from top left
is 327 (= 31 + 10 + 13 + 47 + 65 + 12 + 18 +
6 + 33 + 11 + 20 + 41 + 20)

The chosen least cost path is shown in green.
```

It is not possible to solve this problem using dynamic programming similar to previous problem because here current state depends not only on right and bottom cells but also on left and upper cells. We solve this problem using dijkstra's algorithm. Each cell of grid represents a vertex and neighbor cells adjacent vertices. We do not make an explicit graph

from these cells instead we will use matrix as it is in our dijkstra's algorithm.
In below code Dijkstra' algorithm's implementation using C++ STL is used. The code implemented below is changed to cope with matrix represented implicit graph. Please also see use of dx and dy arrays in below code, these arrays are taken for simplifying the process of visiting neighbor vertices of each cell.

```
 // C++ program to get least cost path in a grid from
// top-left to bottom-right
#include <bits/stdc++.h>
using namespace std;

#define ROW 5
#define COL 5

// structure for information of each cell
struct cell
{
    int x, y;
    int distance;
    cell(int x, int y, int distance) :
        x(x), y(y), distance(distance) {}
};

// Utility method for comparing two cells
bool operator<(const cell& a, const cell& b)
{
    if (a.distance == b.distance)
    {
        if (a.x != b.x)
            return (a.x < b.x);
        else
            return (a.y < b.y);
    }
    return (a.distance < b.distance);
}

// Utility method to check whether a point is
// inside the grid or not
bool isInsideGrid(int i, int j)
{
    return (i >= 0 && i < COL && j >= 0 && j < ROW);
}

// Method returns minimum cost to reach bottom
// right from top left
int shortest(int grid[ROW][COL], int row, int col)
{
    int dis[row][col];
```

```
// initializing distance array by INT_MAX
for (int i = 0; i < row; i++)
    for (int j = 0; j < col; j++)
        dis[i][j] = INT_MAX;

// direction arrays for simplification of getting
// neighbour
int dx[] = {-1, 0, 1, 0};
int dy[] = {0, 1, 0, -1};

set<cell> st;

// insert (0, 0) cell with 0 distance
st.insert(cell(0, 0, 0));

// initialize distance of (0, 0) with its grid value
dis[0][0] = grid[0][0];

// loop for standard dijkstra's algorithm
while (!st.empty())
{
    // get the cell with minimum distance and delete
    // it from the set
    cell k = *st.begin();
    st.erase(st.begin());

    // looping through all neighbours
    for (int i = 0; i < 4; i++)
    {
        int x = k.x + dx[i];
        int y = k.y + dy[i];

        // if not inside boundry, ignore them
        if (!isInsideGrid(x, y))
            continue;

        // If distance from current cell is smaller, then
        // update distance of neighbour cell
        if (dis[x][y] > dis[k.x][k.y] + grid[x][y])
        {
            // If cell is already there in set, then
            // remove its previous entry
            if (dis[x][y] != INT_MAX)
                st.erase(st.find(cell(x, y, dis[x][y])));

            // update the distance and insert new updated
            // cell in set
```

```
                dis[x][y] = dis[k.x][k.y] + grid[x][y];
                st.insert(cell(x, y, dis[x][y]));
            }
        }
    }

    // uncomment below code to print distance
    // of each cell from (0, 0)
    /*
    for (int i = 0; i < row; i++, cout << endl)
        for (int j = 0; j < col; j++)
            cout << dis[i][j] << " ";
    */
    // dis[row - 1][col - 1] will represent final
    // distance of bottom right cell from top left cell
    return dis[row - 1][col - 1];
}

// Driver code to test above methods
int main()
{
    int grid[ROW][COL] =
    {
        31, 100, 65, 12, 18,
        10, 13, 47, 157, 6,
        100, 113, 174, 11, 33,
        88, 124, 41, 20, 140,
        99, 32, 111, 41, 20
    };

    cout << shortest(grid, ROW, COL) << endl;
    return 0;
}
```

Output:

```
327
```

Source

https://www.geeksforgeeks.org/minimum-cost-path-left-right-bottom-moves-allowed/

Chapter 78

Minimum Cost to cut a board into squares

Minimum Cost to cut a board into squares - GeeksforGeeks

A board of length m and width n is given, we need to break this board into m*n squares such that cost of breaking is minimum. cutting cost for each edge will be given for the board. In short we need to choose such a sequence of cutting such that cost is minimized.
Examples:

```
For above board optimal way to cut into square is:
Total minimum cost in above case is 42. It is
evaluated using following steps.

Initial Value : Total_cost = 0
Total_cost = Total_cost + edge_cost * total_pieces

Cost 4 Horizontal cut         Cost = 0 + 4*1 = 4
Cost 4 Vertical cut        Cost = 4 + 4*2 = 12
Cost 3 Vertical cut        Cost = 12 + 3*2 = 18
Cost 2 Horizontal cut        Cost = 18 + 2*3 = 24
Cost 2 Vertical cut        Cost = 24 + 2*3 = 30
Cost 1 Horizontal cut        Cost = 30 + 1*4 = 34
Cost 1 Vertical cut        Cost = 34 + 1*4 = 38
Cost 1 Vertical cut        Cost = 38 + 1*4 = 42
```

This problem can be solved using greedy approach, If total cost is denoted by S, then S = a1w1 + a2w2 … + akwk, where wi is a cost of certain edge cutting and ai is corresponding coefficient, The coefficient ai is determined by the total number of cuts we have competed

using edge wi at the end of the cutting process. Notice that sum of the coefficients are always constant, hence we want to find a distribution of ai obtainable such that S is minimum. To do so **we perform cuts on highest cost edge as early as possible**, which will reach to optimal S. If we encounter several edges having the same cost, we can cut any one of them first.
Below is the solution using above approach, first we sorted the edge cutting costs in reverse order, then we loop in them from higher cost to lower cost building our solution. Each time we choose an edge, counter part count is incremented by 1, which is to be multiplied each time with corresponding edge cutting cost.
Notice below used sort method, sending greater() as 3rd argument to sort method sorts number in non-increasing order, it is predefined function of the library.
C++

```
 //  C++ program to divide a board into m*n squares
#include <bits/stdc++.h>
using namespace std;

// method returns minimum cost to break board into
// m*n squares
int minimumCostOfBreaking(int X[], int Y[], int m, int n)
{
    int res = 0;

    //  sort the horizontal cost in reverse order
    sort(X, X + m, greater<int>());

    //  sort the vertical cost in reverse order
    sort(Y, Y + n, greater<int>());

    //  initialize current width as 1
    int hzntl = 1, vert = 1;

    //  loop untill one or both cost array are processed
    int i = 0, j = 0;
    while (i < m && j < n)
    {
        if (X[i] > Y[j])
        {
            res += X[i] * vert;

            //  increase current horizontal part count by 1
            hzntl++;
            i++;
        }
        else
        {
            res += Y[j] * hzntl;
```

```
            //  increase current vertical part count by 1
            vert++;
            j++;
        }
    }

    // loop for horizontal array, if remains
    int total = 0;
    while (i < m)
        total += X[i++];
    res += total * vert;

    // loop for vertical array, if remains
    total = 0;
    while (j < n)
        total += Y[j++];
    res += total * hzntl;

    return res;
}

//  Driver code to test above methods
int main()
{
    int m = 6, n = 4;
    int X[m-1] = {2, 1, 3, 1, 4};
    int Y[n-1] = {4, 1, 2};
    cout << minimumCostOfBreaking(X, Y, m-1, n-1);
    return 0;
}
```

Java

```
 // Java program to divide a
// board into m*n squares
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // method returns minimum cost to break board into
    // m*n squares
    static int minimumCostOfBreaking(Integer X[], Integer Y[],
                                                    int m, int n)
    {
        int res = 0;

        // sort the horizontal cost in reverse order
```

```
Arrays.sort(X, Collections.reverseOrder());

// sort the vertical cost in reverse order
Arrays.sort(Y, Collections.reverseOrder());

// initialize current width as 1
int hzntl = 1, vert = 1;

// loop untill one or both
// cost array are processed
int i = 0, j = 0;
while (i < m && j < n)
{
    if (X[i] > Y[j])
    {
        res += X[i] * vert;

        // increase current horizontal
        // part count by 1
        hzntl++;
        i++;
    }
    else
    {
        res += Y[j] * hzntl;

        // increase current vertical
        // part count by 1
        vert++;
        j++;
    }
}

// loop for horizontal array,
// if remains
int total = 0;
while (i < m)
    total += X[i++];
res += total * vert;

// loop for vertical array,
// if remains
total = 0;
while (j < n)
    total += Y[j++];
res += total * hzntl;

return res;
```

```
    }

    // Driver program
    public static void main(String arg[])
    {
        int m = 6, n = 4;
        Integer X[] = {2, 1, 3, 1, 4};
        Integer Y[] = {4, 1, 2};
        System.out.print(minimumCostOfBreaking(X, Y, m-1, n-1));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to divide a board into m*n squares

# Method returns minimum cost to
# break board into m*n squares
def minimumCostOfBreaking(X, Y, m, n):

    res = 0

    # sort the horizontal cost in reverse order
    X.sort(reverse = True)

    # sort the vertical cost in reverse order
    Y.sort(reverse = True)

    # initialize current width as 1
    hzntl = 1; vert = 1

    # loop untill one or both
    # cost array are processed
    i = 0; j = 0
    while (i < m and j < n):

        if (X[i] > Y[j]):

            res += X[i] * vert

            # increase current horizontal
            # part count by 1
            hzntl += 1
            i += 1

        else:
```

```
            res += Y[j] * hzntl

            # increase current vertical
            # part count by 1
            vert += 1
            j += 1

    # loop for horizontal array, if remains
    total = 0
    while (i < m):
        total += X[i]
        i += 1
    res += total * vert

    #loop for vertical array, if remains
    total = 0
    while (j < n):
        total += Y[j]
        j += 1
    res += total * hzntl

    return res

# Driver program
m = 6; n = 4
X = [2, 1, 3, 1, 4]
Y = [4, 1, 2]

print(minimumCostOfBreaking(X, Y, m-1, n-1))


# This code is contributed by Anant Agarwal.
```

Output:

```
42
```

Source

https://www.geeksforgeeks.org/minimum-cost-cut-board-squares/

Chapter 79

Minimum Fibonacci terms with sum equal to K

Minimum Fibonacci terms with sum equal to K - GeeksforGeeks

Given a number k, find the required minimum number of Fibonacci terms whose sum equal to k. We can choose a Fibonacci number multiple times.

Examples:

```
Input : k = 4
Output : 2
Fibonacci term added twice that is
2 + 2 = 4.
Other combinations are
1 + 1 + 2.
1 + 1 + 1 + 1
Among all cases first case has the
minimum number of terms = 2.

Input : k = 17
Output : 3
```

We can get any sum using Fibonacci numbers as 1 is a Fibonacci number. For example to get n, we can n times add 1. Here we need to to minimize the count of Fibonacci numbers that contribute to sum. So this problem is basically coin change problemwith coins having fibonacci values. By taking some examples, we can notice that With Fibonacci coin values Greedy approach works.

Firstly we calculate Fibonacci terms till less than or equal to k. then start from the last term and keep subtracting that term from k until k >(nth term). Also along with this keep increasing the count of the number of terms.

When k < (nth Fibonacci term) move to next Fibonacci term which is less than or Equal to k. at last, print the value of count.

The stepwise algorithm is:

```
1. Find all Fibonacci Terms less than or equal to K.
2. Initialize count = 0.
3. j = Index of last calculated Fibonacci Term.
4. while K > 0 do:

    // Greedy step
    count += K / (fibo[j])     // Note that division
                               // is repeated subtraction.
    K = K % (fibo[j])
    j--;
5. Print count.
```

Below is the C++/java implementation of the above approach.
C++

```
 // C++ code to find minimum number of fibonacci
// terms that sum to K.
#include <bits/stdc++.h>
using namespace std;

// Function to calculate Fibonacci Terms
void calcFiboTerms(vector<int>& fiboTerms, int K)
{
    int i = 3, nextTerm;
    fiboTerms.push_back(0);
    fiboTerms.push_back(1);
    fiboTerms.push_back(1);

    // Calculate all Fibonacci terms
    // which are less than or equal to K.
    while (1) {
        nextTerm = fiboTerms[i - 1] + fiboTerms[i - 2];

        // If next term is greater than K
        // do not push it in vector and return.
        if (nextTerm > K)
            return;

        fiboTerms.push_back(nextTerm);
        i++;
    }
}
```

```
// Function to find the minimum number of
// Fibonacci terms having sum equal to K.
int findMinTerms(int K)
{

    // Vector to store Fibonacci terms.
    vector<int> fiboTerms;

    calcFiboTerms(fiboTerms, K);

    int count = 0, j = fiboTerms.size() - 1;

    // Subtract Fibonacci terms from sum K
    // until sum > 0.
    while (K > 0) {

        // Divide sum K by j-th Fibonacci term to find
        // how many terms it contribute in sum.
        count += (K / fiboTerms[j]);
        K %= (fiboTerms[j]);

        j--;
    }

    return count;
}

// driver code
int main()
{
    int K = 17;

    cout << findMinTerms(K);

    return 0;
}
```

Java

```
 // Java code to find minimum number of fibonacci terms
// that sum to k.
import java.util.*;

class GFG
{
    // Function to calaculate Fibonacci Terms
    public static void calcFiboTerms(ArrayList<Integer> fiboterms,
```

```
                                                        int k)
{
    int i = 3, nextTerm = 0;

    fiboterms.add(0);
    fiboterms.add(1);
    fiboterms.add(1);

    // Calculate all Fibonacci terms
    // which are less than or equal to k.
    while(true)
    {
        nextTerm = fiboterms.get(i - 1) + fiboterms.get(i - 2);

        // If next term is greater than k
        // do not add in arraylist and return.
        if(nextTerm>k)
        return;

        fiboterms.add(nextTerm);
        i++;
    }
}

// Function to find the minimum number of
// Fibonacci terms having sum equal to k.
public static int fibMinTerms(int k)
{
    // ArrayList to store Fibonacci terms.
    ArrayList<Integer> fiboterms = new ArrayList<Integer>();
    calcFiboTerms(fiboterms,k);

    int count = 0, j = fiboterms.size() - 1;

    // Subtract Fibonacci terms from sum k
    // until sum > 0.
    while(k > 0)
    {
        // Divide sum k by j-th Fibonacci term to find
        // how many terms it contribute in sum.
        count += (k / fiboterms.get(j));
        k %= (fiboterms.get(j));

        j--;
    }
    return count;
}
```

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

        int k = 17;

        System.out.println(fibMinTerms(k));
    }
}

/* This code is contributed by Akash Singh*/
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/minimum-fibonacci-terms-sum-equal-k/

Chapter 80

Minimum Number of Platforms Required for a Railway/Bus Station

Minimum Number of Platforms Required for a Railway/Bus Station - GeeksforGeeks

Given arrival and departure times of all trains that reach a railway station, find the minimum number of platforms required for the railway station so that no train waits.
We are given two arrays which represent arrival and departure times of trains that stop

Examples:

```
Input:  arr[]  = {9:00,  9:40, 9:50,  11:00, 15:00, 18:00}
        dep[]  = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}
Output: 3
There are at-most three trains at a time (time between 11:00 to 11:20)
```

We need to find the maximum number of trains that are there on the given railway station at a time. A **Simple Solution** is to take every interval one by one and find the number of intervals that overlap with it. Keep track of maximum number of intervals that overlap with an interval. Finally return the maximum value. Time Complexity of this solution is $O(n^2)$.

We can solve the above problem **in O(nLogn) time**. The idea is to consider all events in sorted order. Once we have all events in sorted order, we can trace the number of trains at any time keeping track of trains that have arrived, but not departed.

For example consider the above example.

```
    arr[]  = {9:00,  9:40, 9:50,  11:00, 15:00, 18:00}
```

```
    dep[]  = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}

All events sorted by time.
Total platforms at any time can be obtained by subtracting total
departures from total arrivals by that time.
 Time     Event Type     Total Platforms Needed at this Time
 9:00       Arrival                  1
 9:10       Departure                0
 9:40       Arrival                  1
 9:50       Arrival                  2
 11:00      Arrival                  3
 11:20      Departure                2
 11:30      Departure                1
 12:00      Departure                0
 15:00      Arrival                  1
 18:00      Arrival                  2
 19:00      Departure                1
 20:00      Departure                0

Minimum Platforms needed on railway station = Maximum platforms
                                                needed at any time
                                            = 3
```

Following is the implementation of above approach. Note that the implementation doesn't create a single sorted list of all events, rather it individually sorts arr[] and dep[] arrays, and then uses merge process of merge sort to process them together as a single sorted array.
Note : This approach assumes that trains are arriving and departing on same date.

C++

```
 // Program to find minimum number of platforms
// required on a railway station
#include<iostream>
#include<algorithm>

using namespace std;

// Returns minimum number of platforms reqquired
int findPlatform(int arr[], int dep[], int n)
{
   // Sort arrival and departure arrays
   sort(arr, arr+n);
   sort(dep, dep+n);

   // plat_needed indicates number of platforms
   // needed at a time
   int plat_needed = 1, result = 1;
   int i = 1, j = 0;
```

```
   // Similar to merge in merge sort to process
   // all events in sorted order
   while (i < n && j < n)
   {
      // If next event in sorted order is arrival,
      // increment count of platforms needed
      if (arr[i] <= dep[j])
      {
          plat_needed++;
          i++;

          // Update result if needed
          if (plat_needed > result)
              result = plat_needed;
      }

      // Else decrement count of platforms needed
      else
      {
          plat_needed--;
          j++;
      }
   }

   return result;
}

// Driver program to test methods of graph class
int main()
{
    int arr[] = {900, 940, 950, 1100, 1500, 1800};
    int dep[] = {910, 1200, 1120, 1130, 1900, 2000};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum Number of Platforms Required = "
         << findPlatform(arr, dep, n);
    return 0;
}
```

Java

```
 // Program to find minimum number of platforms

import java.util.*;

class GFG {

// Returns minimum number of platforms reqquired
```

```
static int findPlatform(int arr[], int dep[], int n)
{
   // Sort arrival and departure arrays
   Arrays.sort(arr);
   Arrays.sort(dep);

   // plat_needed indicates number of platforms
   // needed at a time
   int plat_needed = 1, result = 1;
   int i = 1, j = 0;

   // Similar to merge in merge sort to process
   // all events in sorted order
   while (i < n && j < n)
   {
      // If next event in sorted order is arrival,
      // increment count of platforms needed
      if (arr[i] <= dep[j])
      {
          plat_needed++;
          i++;

          // Update result if needed
          if (plat_needed > result)
              result = plat_needed;
      }

      // Else decrement count of platforms needed
      else
      {
          plat_needed--;
          j++;
      }
   }

   return result;
}

// Driver program to test methods of graph class
public static void main(String[] args)
{
    int arr[] = {900, 940, 950, 1100, 1500, 1800};
    int dep[] = {910, 1200, 1120, 1130, 1900, 2000};
    int n = arr.length;
    System.out.println("Minimum Number of Platforms Required = "
                       + findPlatform(arr, dep, n));
}
}
```

Python3

```
 # Program to find minimum
# number of platforms
# required on a railway
# station

# Returns minimum number
# of platforms reqquired
def findPlatform(arr,dep,n):

    # Sort arrival and
    # departure arrays
    arr.sort()
    dep.sort()

    # plat_needed indicates
    # number of platforms
    # needed at a time
    plat_needed = 1
    result = 1
    i = 1
    j = 0

    # Similar to merge in
    # merge sort to process
    # all events in sorted order
    while (i < n and j < n):

        # If next event in sorted
        # order is arrival,
        # increment count of
        # platforms needed
        if (arr[i] < dep[j]):

            plat_needed+=1
            i+=1

            # Update result if needed
            if (plat_needed > result):
                result = plat_needed


        # Else decrement count
        # of platforms needed
        else:

            plat_needed-=1
```

```
            j+=1

    return result

# driver code

arr = [900, 940, 950, 1100, 1500, 1800]
dep = [910, 1200, 1120, 1130, 1900, 2000]
n = len(arr)

print("Minimum Number of Platforms Required = ",
        findPlatform(arr, dep, n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find minimum number
// of platforms
using System;

class GFG {

    // Returns minimum number of platforms
    // reqquired
    static int findPlatform(int []arr,
                        int []dep, int n)
    {

        // Sort arrival and departure arrays
        Array.Sort(arr);
        Array.Sort(dep);

        // plat_needed indicates number of
        // platforms needed at a time
        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        // Similar to merge in merge sort
        // to process all events in sorted
        // order
        while (i < n && j < n)
        {

            // If next event in sorted order
            // is arrival, increment count
            // of platforms needed
```

```
            if (arr[i] <= dep[j])
            {
                plat_needed++;
                i++;

                // Update result if needed
                if (plat_needed > result)
                    result = plat_needed;
            }

            // Else decrement count of
            // platforms needed
            else
            {
                plat_needed--;
                j++;
            }
        }

        return result;
    }

    // Driver program to test methods of
    // graph class
    public static void Main()
    {
        int []arr = {900, 940, 950, 1100,
                                1500, 1800};
        int []dep = {910, 1200, 1120, 1130,
                                1900, 2000};
        int n = arr.Length;
        Console.Write("Minimum Number of "
                  + " Platforms Required = "
                + findPlatform(arr, dep, n));
    }
}

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

PHP

```
 <?php
// PHP Program to find minimum number
// of platforms  required on a railway
// station

// Returns minimum number of
// platforms reqquired
```

```
function findPlatform($arr, $dep, $n)
{

    // Sort arrival and
    // departure arrays
    sort($arr);
    sort($dep);

    // plat_needed indicates
    // number of platforms
    // needed at a time
    $plat_needed = 1;
    $result = 1;
    $i = 1;
    $j = 0;

    // Similar to merge in
    // merge sort to process
    // all events in sorted order
    while ($i < $n and $j < $n)
    {

        // If next event in sorted
        // order is arrival, increment
        // count of platforms needed
        if ($arr[$i] <= $dep[$j])
        {
            $plat_needed++;
            $i++;

            // Update result if needed
            if ($plat_needed > $result)
                $result = $plat_needed;
        }

        // Else decrement count
        // of platforms needed
        else
        {
            $plat_needed--;
            $j++;
        }
    }

    return $result;
}

    // Driver Code
```

```
    $arr = array(900, 940, 950, 1100, 1500, 1800);
    $dep = array(910, 1200, 1120, 1130, 1900, 2000);
    $n = count($arr);
    echo "Minimum Number of Platforms Required = "
                   , findPlatform($arr, $dep, $n);

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

Output:

```
Minimum Number of Platforms Required = 3
```

Algorithmic Paradigm: Dynamic Programming

Time Complexity: O(nLogn), assuming that a O(nLogn) sorting algorithm for sorting arr[] and dep[].

Minimum Number of Platforms Required for a Railway/Bus Station Set 2 (Map based approach)

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

Improved By : siddhartha33, nitin mittal, vt_m, harrypotter0

Source

https://www.geeksforgeeks.org/minimum-number-platforms-required-railwaybus-station/

Chapter 81

Minimum Swaps for Bracket Balancing

Minimum Swaps for Bracket Balancing - GeeksforGeeks

You are given a string of 2N characters consisting of N '[' brackets and N ']' brackets. A string is considered balanced if it can be represented in the for S2[S1] where S1 and S2 are balanced strings. We can make an unbalanced string balanced by swapping adjacent characters. Calculate the minimum number of swaps necessary to make a string balanced.

Examples:

```
Input  : []][][
Output : 2
First swap: Position 3 and 4
[][]][
Second swap: Position 5 and 6
[][][]

Input  : [[][]]
Output : 0
String is already balanced.
```

We can solve this problem using greedy strategies. If the first X characters form a balanced string, we can neglect these characters and continue on. If we encounter a ']' before the required '[', then we must start swapping elements to balance the string.

Naive Approach
Initialize sum = 0 where **sum** stores result. Go through the string maintaining a **count** of the number of '[' brackets encountered. Reduce this count when we encounter a ']' character. If the count hits negative, then we must start balancing the string.
Let index 'i' represents the position we are at. We now move forward to the next '[' at

index j. Increase sum by j – i. Move the '[' at position j, to position i, and shift all other characters to the right. Set the count back to 0 and continue traversing the string. At the end 'sum' will have the required value.

Time Complexity = O(N^2)
Extra Space = O(1)

Optimized approach
We can initially go through the string and store the positions of '[' in a vector say '**pos**'. Initialize 'p' to 0. We shall use p to traverse the vector 'pos'. Similar to the naive approach, we maintain a count of encountered '[' brackets. When we encounter a '[' we increase the count, and increase 'p' by 1. When we encounter a ']' we decrease the count. If the count ever goes negative, this means we must start swapping. The element pos[p] tells us the index of the next '['. We increase the sum by pos[p] – i, where i is the current index. We can swap the elements in the current index and pos[p] and reset count to 0.

Since we have converted a step that was O(N) in the naive approach, to an O(1) step, our new time complexity reduces.
Time Complexity = O(N)
Extra Space = O(N)

```
 // Program to count swaps required to balance string
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Function to calculate swaps required
long swapCount(string s)
{
    // Keep track of '['
    vector<int> pos;
    for (int i = 0; i < s.length(); ++i)
        if (s[i] == '[')
            pos.push_back(i);

    int count = 0; // To count number of encountered '['
    int p = 0;  // To track position of next '[' in pos
    long sum = 0; // To store result

    for (int i = 0; i < s.length(); ++i)
    {
        // Increment count and move p to next position
        if (s[i] == '[')
        {
            ++count;
            ++p;
        }
        else if (s[i] == ']')
```

```
            --count;

        // We have encountered an unbalanced part of string
        if (count < 0)
        {
            // Increment sum by number of swaps required
            // i.e. position of next '[' - current position
            sum += pos[p] - i;
            swap(s[i], s[pos[p]]);
            ++p;

            // Reset count to 1
            count = 1;
        }
    }
    return sum;
}

// Driver code
int main()
{
    string s = "[]][][";
    cout << swapCount(s) << "\n";

    s = "[[][]]";
    cout << swapCount(s) << "\n";
    return 0;
}
```

Output:

```
2
0
```

Source

https://www.geeksforgeeks.org/minimum-swaps-bracket-balancing/

Chapter 82

Minimum cost for acquiring all coins with k extra coins allowed with every coin

Minimum cost for acquiring all coins with k extra coins allowed with every coin - GeeksforGeeks

You are given a list of N coins of different denominations. you can pay an amount equivalent to any 1 coin and can acquire that coin. In addition, once you have paid for a coin, we can choose at most K more coins and can acquire those for free. The task is to find the minimum amount required to acquire all the N coins for a given value of K.

Examples :

```
Input : coin[] = {100, 20, 50, 10, 2, 5},
        k = 3
Output : 7

Input : coin[] = {1, 2, 5, 10, 20, 50},
        k = 3
Output : 3
```

As per question, we can see that at a cost of 1 coin, we can acquire at most K+1 coins. Therefore, in order to acquire all the n coins, we will be choosing ceil(n/(k+1)) coins and the cost of choosing coins will be minimum if we choose smallest ceil(n/(k+1)) (Greedy approach). Smallest ceil(n/(k+1)) coins can be found by simply sorting all the N values in increasing order.
If we should check for time complexity (n log n) is for sorting element and (k) is for adding the total amount. So, finally Time Complexity : O(n log n).

C++

```
 // C++ program to acquire all n coins
#include<bits/stdc++.h>
using namespace std;

// function to calculate min cost
int minCost(int coin[], int n, int k)
{
    // sort the coins value
    sort(coin, coin + n);

    // calculate no. of
    // coins needed
    int coins_needed = ceil(1.0 * n /
                            (k + 1));

    // calculate sum of
    // all selected coins
    int ans = 0;
    for (int i = 0; i <= coins_needed - 1;
                                       i++)
        ans += coin[i];

    return ans;
}

// Driver Code
int main()
{
    int coin[] = {8, 5, 3, 10,
                  2, 1, 15, 25};
    int n = sizeof(coin) / sizeof(coin[0]);
    int k = 3;
    cout << minCost(coin, n, k);
    return 0;
}
```

Java

```
 // Java program to acquire
// all n coins
import java.util.Arrays;

class GFG
{

    // function to calculate min cost
    static int minCost(int coin[],
                       int n, int k)
```

```
    {

        // sort the coins value
        Arrays.sort(coin);

        // calculate no. of
        // coins needed
        int coins_needed = (int)Math.ceil(1.0 *
                                  n / (k + 1));

        // calculate sum of
        // all selected coins
        int ans = 0;

        for (int i = 0; i <= coins_needed - 1;
                                           i++)
            ans += coin[i];

        return ans;
    }

    // Driver code
    public static void main(String arg[])
    {
        int coin[] = { 8, 5, 3, 10,
                       2, 1, 15, 25 };
        int n = coin.length;
        int k = 3;

        System.out.print(minCost(coin, n, k));
    }
}

// This code is contributed
// by Anant Agarwal.
```

Python3

```
 # Python program to
# acquire all n coins

import math

# function to calculate min cost
def minCost(coin, n, k):

    # sort the coins value
    coin.sort()
```

```
    # calculate no. of
    # coins needed
    coins_needed = math.ceil(1.0 * n //
                             (k + 1));

    # calculate sum of all
    # selected coins
    ans = 0
    for i in range(coins_needed - 1 + 1):
        ans += coin[i]

    return ans

# Driver code
coin = [8, 5, 3, 10,
        2, 1, 15, 25]
n = len(coin)
k = 3

print(minCost(coin, n, k))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to acquire all n coins
using System;

class GFG
{

    // function to calculate min cost
    static int minCost(int []coin,
                       int n, int k)
    {
        // sort the coins value
        Array.Sort(coin);

        // calculate no. of coins needed
        int coins_needed = (int)Math.Ceiling(1.0 *
                                    n / (k + 1));

        // calculate sum of
        // all selected coins
        int ans = 0;
```

```
        for (int i = 0; i <= coins_needed - 1; i++)
            ans += coin[i];

        return ans;
    }

    // Driver code
    public static void Main()
    {
        int []coin = {8, 5, 3, 10,
                      2, 1, 15, 25};
        int n = coin.Length;
        int k = 3;

        // Function calling
        Console.Write(minCost(coin, n, k));
    }
}

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

PHP

```
 <?php
// PHP program to acquire all n coins


// function to calculate min cost
function minCost($coin, $n, $k)
{
    // sort the coins value
    sort($coin); sort($coin,$n);

    // calculate no. of coins needed
    $coins_needed = ceil(1.0 * $n / ($k + 1));

    // calculate sum of
    // all selected coins
    $ans = 0;
    for ($i = 0; $i <= $coins_needed - 1; $i++)
        $ans += $coin[$i];

    return $ans;
}

// Driver Code
{
```

```
    $coin = array(8, 5, 3, 10,
                  2, 1, 15, 25);
    $n = sizeof($coin) / sizeof($coin[0]);
    $k = 3;
    echo minCost($coin, $n, $k);
    return 0;
}

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

Output :

```
3
```

Note that there are more efficient approaches to find given number of smallest values. For example, method 6 of m largest(or smallest) elements in an array can find m'th smallest element in (n-m) Log m + m Log m).

How to handle multiple queries for a single predefined array?
In the case, if you are asked to find the above answer for many different values of K, you have to compute it fast and our time complexity got increased as per number of queries for k. For the purpose to serve, we can maintain a prefix sum array after sorting all the N values and can answer queries easily and quickly.
Suppose
C++

```
 // C++ program to acquire all
// n coins at minimum cost
// with multiple values of k.
#include<bits/stdc++.h>
using namespace std;

// Converts coin[] to prefix sum array
void preprocess(int coin[], int n)
{
    // sort the coins value
    sort(coin, coin + n);

    // Maintain prefix sum array
    for (int i = 1; i <= n - 1; i++)
        coin[i] += coin[i - 1];
}

// Function to calculate min
// cost when we can get k extra
```

```
// coins after paying cost of one.
int minCost(int coin[], int n, int k)
{
    // calculate no. of coins needed
    int coins_needed = ceil(1.0 * n / (k + 1));

    // return sum of from prefix array
    return coin[coins_needed - 1];
}

// Driver Code
int main()
{
    int coin[] = {8, 5, 3, 10,
                  2, 1, 15, 25};
    int n = sizeof(coin) / sizeof(coin[0]);
    preprocess(coin, n);
    int k = 3;
    cout << minCost(coin, n, k) << endl;
    k = 7;
    cout << minCost(coin, n, k) << endl;
    return 0;
}
```

Java

```
 // C# program to acquire all n coins at
// minimum cost with multiple values of k.
import java .io.*;
import java.util.Arrays;

public class GFG {

    // Converts coin[] to prefix sum array
    static void preprocess(int []coin, int n)
    {

        // sort the coins value
        Arrays.sort(coin);

        // Maintain prefix sum array
        for (int i = 1; i <= n - 1; i++)
            coin[i] += coin[i - 1];
    }

    // Function to calculate min cost when we
    // can get k extra coins after paying
    // cost of one.
```

```
    static int minCost(int []coin, int n, int k)
    {

        // calculate no. of coins needed
        int coins_needed =(int) Math.ceil(1.0
                                * n / (k + 1));

        // return sum of from prefix array
        return coin[coins_needed - 1];
    }

    // Driver Code
    static public void main (String[] args)
    {
        int []coin = {8, 5, 3, 10, 2, 1, 15, 25};
        int n = coin.length;

        preprocess(coin, n);

        int k = 3;
        System.out.println(minCost(coin, n, k));

        k = 7;
        System.out.println( minCost(coin, n, k));
    }
}

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

C#

```
 // C# program to acquire all n coins at
// minimum cost with multiple values of k.
using System;

public class GFG {

    // Converts coin[] to prefix sum array
    static void preprocess(int []coin, int n)
    {

        // sort the coins value
        Array.Sort(coin);

        // Maintain prefix sum array
        for (int i = 1; i <= n - 1; i++)
            coin[i] += coin[i - 1];
    }
```

```
    // Function to calculate min cost when we
    // can get k extra coins after paying
    // cost of one.
    static int minCost(int []coin, int n, int k)
    {

        // calculate no. of coins needed
        int coins_needed =(int) Math.Ceiling(1.0
                                * n / (k + 1));

        // return sum of from prefix array
        return coin[coins_needed - 1];
    }

    // Driver Code
    static public void Main ()
    {
        int []coin = {8, 5, 3, 10, 2, 1, 15, 25};
        int n = coin.Length;

        preprocess(coin, n);

        int k = 3;
        Console.WriteLine(minCost(coin, n, k));

        k = 7;
        Console.WriteLine( minCost(coin, n, k));
    }
}

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

Output :

```
3
1
```

After preprocessing, every query for a k takes O(1) time.

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/minimum-cost-for-acquiring-all-coins-with-k-extra-coins-allowed-with-every-coin/

Chapter 83

Minimum cost to connect all cities

Minimum cost to connect all cities - GeeksforGeeks

There are n cities and there are roads in between some of the cities. Somehow all the roads are damaged simultaneously. We have to repair the roads to connect the cities again. There is a fixed cost to repair a particular road. Find out the minimum cost to connect all the cities by repairing roads. Input is in matrix(city) form, if city[i][j] = 0 then there is not any road between city i and city j, if city[i][j] = a > 0 then the cost to rebuild the path between city i and city j is a. Print out the minimum cost to connect all the cities.
It is sure that all the cities were connected before the roads were damaged.

Examples:

```
Input : {{0, 1, 2, 3, 4},
         {1, 0, 5, 0, 7},
         {2, 5, 0, 6, 0},
         {3, 0, 6, 0, 0},
         {4, 7, 0, 0, 0}};
Output : 10
```

```
Input : {{0, 1, 1, 100, 0, 0},
         {1, 0, 1, 0, 0, 0},
         {1, 1, 0, 0, 0, 0},
         {100, 0, 0, 0, 2, 2},
         {0, 0, 0, 2, 0, 2},
         {0, 0, 0, 2, 2, 0}};
Output : 106
```

Method: Here we have to connect all the cities by path which will cost us least. The way to do that is to find out the Minimum Spanning Tree(MST) of the map of the cities(i.e.

each city is a node of the graph and all the damaged roads between cities are edges). And the total cost is the addition of the path edge values in the Minimum Spanning Tree.

Prerequisite: MST Prim's Algorithm

C++

\

```
 // C++ code to find out minimum cost
// path to connect all the cities
#include <iostream>
#include <limits>
#include <vector>

using namespace std;

// Function to find out minimum valued node
// among the nodes which are not yet included in MST
int minnode(int n, int keyval[], bool mstset[]) {
  int mini = numeric_limits<int>::max();
  int mini_index;

  // Loop through all the values of the nodes
  // which are not yet included in MST and find
  // the minimum valued one.
  for (int i = 0; i < n; i++) {
    if (mstset[i] == false && keyval[i] < mini) {
      mini = keyval[i], mini_index = i;
    }
  }
  return mini_index;
}

// Function to find out the MST and
// the cost of the MST.
void findcost(int n, vector<vector<int>> city) {

  // Array to store the parent node of a
  // particular node.
  int parent[n];

  // Array to store key value of each node.
  int keyval[n];

  // Boolean Array to hold bool values whether
  // a node is included in MST or not.
  bool mstset[n];
```

```
  // Set all the key values to infinite and
  // none of the nodes is included in MST.
  for (int i = 0; i < n; i++) {
    keyval[i] = numeric_limits<int>::max();
    mstset[i] = false;
  }

  // Start to find the MST from node 0.
  // Parent of node 0 is none so set -1.
  // key value or minimum cost to reach
  // 0th node from 0th node is 0.
  parent[0] = -1;
  keyval[0] = 0;

  // Find the rest n-1 nodes of MST.
  for (int i = 0; i < n - 1; i++) {

    // First find out the minimum node
    // among the nodes which are not yet
    // included in MST.
    int u = minnode(n, keyval, mstset);

    // Now the uth node is included in MST.
    mstset[u] = true;

    // Update the values of neighbor
    // nodes of u which are not yet
    // included in MST.
    for (int v = 0; v < n; v++) {

      if (city[u][v] && mstset[v] == false &&
          city[u][v] < keyval[v]) {
        keyval[v] = city[u][v];
        parent[v] = u;
      }
    }
  }

  // Find out the cost by adding
  // the edge values of MST.
  int cost = 0;
  for (int i = 1; i < n; i++)
    cost += city[parent[i]][i];
  cout << cost << endl;
}

// Utility Program:
int main() {
```

```
  // Input 1
  int n1 = 5;
  vector<vector<int>> city1 = {{0, 1, 2, 3, 4},
                               {1, 0, 5, 0, 7},
                               {2, 5, 0, 6, 0},
                               {3, 0, 6, 0, 0},
                               {4, 7, 0, 0, 0}};
  findcost(n1, city1);

  // Input 2
  int n2 = 6;
  vector<vector<int>> city2 = {{0, 1, 1, 100, 0, 0},
                               {1, 0, 1, 0, 0, 0},
                               {1, 1, 0, 0, 0, 0},
                               {100, 0, 0, 0, 2, 2},
                               {0, 0, 0, 2, 0, 2},
                               {0, 0, 0, 2, 2, 0}};
  findcost(n2, city2);

  return 0;
}
```

Output:

```
10
106
```

Complexity: The outer loop(i.e. the loop to add new node to MST) runs n times and in each iteration of the loop it takes O(n) time to find the minnode and O(n) time to update the neighboring nodes of u-th node. Hence the overall complexity is $O(n^2)$

Source

https://www.geeksforgeeks.org/minimum-cost-connect-cities/

Chapter 84

Minimum cost to make array size 1 by removing larger of pairs

Minimum cost to make array size 1 by removing larger of pairs - GeeksforGeeks

Given an array of n integers. We need to reduce size of array to one. We are allowed to select a pair of integers and remove the larger one of these two. This decreases the array size by 1. Cost of this operation is equal to value of smaller one. Find out minimum sum of costs of operations needed to convert the array into a single element.

Examples:

```
Input: 4 3 2
Output: 4
Explanation:
Choose (4, 2) so 4 is removed, new array
= {2, 3}. Now choose (2, 3) so 3 is removed.
So total cost = 2 + 2 = 4

Input: 3 4
Output: 3
Explanation: choose 3, 4, so cost is 3.
```

The idea is to always pick minimum value as part of the pair and remove larger value. This minimizes cost of reducing array to size 1.

Below is the implementation of the above approach:
CPP

```
// CPP program to find minimum cost to
```

```
// reduce array size to 1,
#include <bits/stdc++.h>
using namespace std;

// function to calculate the minimum cost
int cost(int a[], int n)
{
    // Minimum cost is n-1 multiplied with
    // minimum element.
    return (n - 1) * (*min_element(a, a + n));
}

// driver program to test the above function.
int main()
{
    int a[] = { 4, 3, 2 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << cost(a, n) << endl;
    return 0;
}
```

Java

```
 // Java program to find minimum cost
// to reduce array size to 1,
import java.lang.*;

public class GFG {

    // function to calculate the
    // minimum cost
    static int cost(int []a, int n)
    {
        int min = a[0];

        // find the minimum using
        // for loop
        for(int i = 1; i< a.length; i++)
        {
            if (a[i] < min)
                min = a[i];
        }

        // Minimum cost is n-1 multiplied
        // with minimum element.
        return (n - 1) * min;
    }
```

```
    // driver program to test the
    // above function.
    static public void main (String[] args)
    {

        int []a = { 4, 3, 2 };
        int n = a.length;

        System.out.println(cost(a, n));
    }
}

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

Python3

```
 # Python program to find minimum
# cost to reduce array size to 1

# function to calculate the
# minimum cost
def cost(a, n):

    # Minimum cost is n-1 multiplied
    # with minimum element.
    return ( (n - 1) * min(a) )


# driver code
a = [ 4, 3, 2 ]
n = len(a)
print(cost(a, n))

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

C#

```
 // C# program to find minimum cost to
// reduce array size to 1,
using System;
using System.Linq;

public class GFG {

    // function to calculate the minimum cost
    static int cost(int []a, int n)
```

```
    {

        // Minimum cost is n-1 multiplied with
        // minimum element.
        return (n - 1) * a.Min();
    }

    // driver program to test the above function.
    static public void Main (){

        int []a = { 4, 3, 2 };
        int n = a.Length;

        Console.WriteLine(cost(a, n));
    }
}

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

PHP

```
 <?php
// PHP program to find minimum cost to
// reduce array size to 1,

// function to calculate
// the minimum cost
function cost($a, $n)
{

    // Minimum cost is n-1
    // multiplied with
    // minimum element.
    return ($n - 1) * (min($a));
}

    // Driver Code
    $a = array(4, 3, 2);
    $n = count($a);
    echo cost($a, $n);

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

Output:

```
4
```

Time Complexity : O(n)

Improved By : parashar, vt_m

Source

https://www.geeksforgeeks.org/minimum-cost-make-array-size-1-removing-larger-pairs/

Chapter 85

Minimum cost to process m tasks where switching costs

Minimum cost to process m tasks where switching costs - GeeksforGeeks

There are n cores of processor. Each core can process a task at a time. Different cores can process different tasks simultaneously without affecting others. Suppose, there are m tasks in queue and the processor has to process these m tasks. Again, these m tasks are not all of similar type. The type of task is denoted by a number. So, m tasks are represented as m consecutive numbers, same number represents same type of task, like 1 represents task of type 1, 2 for type 2 task and so on.
Initially, all the cores are free. It takes one unit of cost to start a type of task in a core, which is currently not running in that core. One unit cost will be charged at the starting to start tasks on each core. As an example, a core is running type 3 task and if we assign type 3 task again in that core, then cost for this assignment will be zero. But, if we assign type 2 task then cost for this assignment will be one unit. A core keeps processing a task until next task is assigned to that core.

Example :

```
Input : n = 3, m = 6, arr = {1, 2, 1, 3, 4, 1}
Output : 4

Input : n = 2, m = 6, arr = {1, 2, 1, 3, 2, 1}
Output : 4
Explanation :

Input : n = 3, m = 31,
arr = {7, 11, 17, 10, 7, 10, 2, 9, 2, 18, 8, 10, 20, 10, 3, 20,
       17, 17, 17, 1, 15, 10, 8, 3, 3, 18, 13, 2, 10, 10, 11}
Output : 18
```

Explanation (for 1st sample I/O) : Here total number of cores are 3. Let, A, B and C. First assign task of type 1 in any of the cores -> cost 1 unit. States: A – 1, B – None, C – None. Assign task of type 2 in any of the rest 2 cores -> cost 1 unit. States : A – 1, B – 2, C – None. Then assign task of type 1 in that core where task of type 1 is ongoing -> cost 0 unit. States : A – 1, B – 2, C – None.
Assign task of type 3 in the free core -> cost 1 unit. States : A – 1, B – 2, C – 3.
Now, all the cores are running a task. So we have to assign task of type 4 in one of these cores. Let's load it in the core B, where previously type 2 task was going on -> cost 1 unit. States: A – 1, B – 4, C – 3. Now, load the type 1 task in the core A, where type 1 task is running -> cost 0 unit. States: A – 1, B – 4, C – 3. Hence, total cost = 1 + 1 + 0 + 1 + 1 + 0 = 4 u
units.

Explanation 2 (for 2nd sample I/O) : Total number of cores are 2. Let A and B. First process task of type 1 in any of the cores -> cost 1 unit. States: A – 1, B – None. Process task of type 2 in any of the rest 2 cores -> cost 1 unit. States: A – 1, B – 2. Then process task of type 1 in that core where task of type 1 is ongoing -> cost 0 unit. States : A – 1, B – 2. Now, let's assign task of type 3 to core A -> cost 1 unit. States : A – 3, B – 2. Now, assign type 2 task in core B, where already type 2 task is going on -> cost 0 unit. States : A – 3, B – 2. Hence, total cost = 1 + 1 + 0 + 1 + 1 = 4 unit. Last assign type 1 task in any of the cores(say A) -> cost 1 unit. States : A – 1, B – 2. Hence, total cost = 1 + 1 + 0 + 1 + 0 + 1 = 4 units.

Approach : Dividing this problem into two cases :
First one is when same type of task is currently running in one of the cores. Then, just assign the upcoming task to that core. For example 1, at i = 3(third task), when type 1 task comes then we can assign that task to that core where previously type 1 task was going on. Cost of this is 0 unit.
Second case is when the same type of task is not running in any of the cores. Then, there may be two possibilities. Sub-case 1 is, if there is at least one free core then assign the upcoming task to that core.

Sub-case 2 is, if there are no free core then we have to stop processing one type of task, free up a core and assign the upcoming task in that core such that the cost in future becomes minimum. To minimize cost we should stop one type of task which will never occur again in future. If every ongoing task reoccurs at least once in future, then we should stop that task which will reoccur last among all the currently ongoing tasks(it is a greedy approach and will task).
For example 2 : at i = 4(fourth task), type 3 task, currently ongoing type 1 and type 2 tasks in two cores, we can stop task type 1 or task type 2 to start task type 3. But we will stop task type 1 as type 1 task rehappens after type two task.

```
 // C++ Program to fid out minimum
// cost to process m tasks
#include <bits/stdc++.h>

using namespace std;

// Function to find out the farthest
```

```
// position where one of the currently
// ongoing tasks will rehappen.
int find(vector<int> arr, int pos,
         int m, vector<bool> isRunning)
{

    // Iterate form last to current
    // position and find where the
    // task will happen next.
    vector<int> d(m + 1, 0);
    for (int i = m; i > pos; i--)
    {
        if (isRunning[arr[i]])
            d[arr[i]] = i;
    }

    // Find out maximum of all these
    // positions and it is the
    // farthest position.
    int maxipos = 0;
    for (int ele : d)
        maxipos = max(ele, maxipos);

    return maxipos;
}

// Function to find out minimum cost to
// process all the tasks
int mincost(int n, int m, vector<int> arr)
{

    // freqarr[i][j] denotes the frequency
    // of type j task after position i
    // like in array {1, 2, 1, 3, 2, 1}
    // frequency of type 1 task after
    // position 0 is 2. So, for this
    // array freqarr[0][1] = 2. Here,
    // i can range in between 0 to m-1 and
    // j can range in between 0 to m(though
    // there is not any type 0 task).
    vector<vector<int> > freqarr(m);

    // Fill up the freqarr vector from
    // last to first. After m-1 th position
    // frequency of all type of tasks will be
    // 0. Then at m-2 th position only frequency
    // of arr[m-1] type task will be increased
    // by 1. Again, in m-3 th position only
```

```
// frequency of type arr[m-2] task will
// be increased by 1 and so on.
vector<int> newvec(m + 1, 0);
freqarr[m - 1] = newvec;
for (int i = m - 2; i >= 0; i--)
{
    vector<int> nv;
    nv = freqarr[i + 1];
    nv[arr[i + 1]] += 1;
    freqarr[i] = nv;
}

// isRunning[i] denotes whether type i
// task is currently running in one
// of the cores.
// At the beginning no tasks are
// running so all values are false.
vector<bool> isRunning(m + 1, false);

// cost denotes total cost to assign tasks
int cost = 0;

// truecount denotes number of occupied cores
int truecount = 0;

// iterate through each task and find the
// total cost.
for (int i = 0; i < m; i++) {

    // ele denotes type of task.
    int ele = arr[i];

    // Case 1: if smae type of task is
    // currently running cost for this
    // is 0.
    if (isRunning[ele] == true)
        continue;

    // Case 2: same type of task is not
    // currently running.
    else {

        // Subcase 1: if there is at least
        // one free core then assign this task
        // to that core at a cost of 1 unit.
        if (truecount < n) {
            cost += 1;
            truecount += 1;
```

```
    isRunning[ele] = true;
}

// Subcase 2: No core is free
else {

    // here we will first find the minimum
    // frequency among all the ongoing tasks
    // in future.
    // If the minimum frequency is 0 then
    // there is at least one ongoing task
    // which will not occur again. Then we just
    // stop tha task.
    // If minimum frequency is >0 then, all the
    // tasks will occur at least once in future.
    // we have to find out which of these will
    // rehappen last among the all ongoing tasks.

    // set minimum frequency to a big number
    int mini = 100000;

    // set index of minimum frequency task to 0.
    int miniind = 0;

    // find the minimum frequency task type(miniind)
    // and frequency(mini).
    for (int j = 1; j <= m; j++) {
        if (isRunning[j] && mini > freqarr[i][j]) {
            mini = freqarr[i][j];
            miniind = j;
        }
    }

    // If minimum frequency is zero then just stop
    // the task and start the present task in that
    // core. Cost for this is 1 unit.
    if (mini == 0) {
        isRunning[miniind] = false;
        isRunning[ele] = true;
        cost += 1;
    }

    // If minimum frequency is nonzero then find
    // the farthest position where one of the
    // ongoing task will rehappen.
    // Stop that task and start present task
    // in that core.
    else {
```

```
                    // find out the farthest position using
                    // find function
                    int farpos = find(arr, i, m, isRunning);
                    isRunning[arr[farpos]] = false;
                    isRunning[ele] = true;
                    cost += 1;
                }
            }
        }
    }
    // return total cost
    return cost;
}

// Driver Program
int main()
{
    // Test case 1
    int n1 = 3;
    int m1 = 6;
    vector<int> arr1{ 1, 2, 1, 3, 4, 1 };
    cout << mincost(n1, m1, arr1) << endl;

    // Test case 2
    int n2 = 2;
    int m2 = 6;
    vector<int> arr2{ 1, 2, 1, 3, 2, 1 };
    cout << mincost(n2, m2, arr2) << endl;

    // Test case 3
    int n3 = 3;
    int m3 = 31;
    vector<int> arr3{ 7, 11, 17, 10, 7, 10, 2, 9,
                      2, 18, 8, 10, 20, 10, 3, 20,
                      17, 17, 17, 1, 15, 10, 8, 3,
                       3, 18, 13, 2, 10, 10, 11 };
    cout << mincost(n3, m3, arr3) << endl;

    return 0;
}
```

Output:

```
4
4
18
```

Time Complexity : O(m^2)
Space Complexity : O(m^2), (to store the freqarr)

Source

https://www.geeksforgeeks.org/minimum-cost-to-process-m-tasks-where-switching-costs/

Chapter 86

Minimum difference between groups of size two

Minimum difference between groups of size two - GeeksforGeeks

Given an array of even number of elements, form groups of 2 using these array elements such that the difference between the group with highest sum and the one with lowest sum is minimum.

Note: An element can be a part of one group only and it has to be a part of at least 1 group.

Examples:

```
Input : arr[] = {2, 6, 4, 3}
Output : 1
Groups formed will be (2, 6) and (4, 3),
the difference between highest sum group
(2, 6) i.e 8 and lowest sum group (3, 4)
i.e 7 is 1.

Input : arr[] = {11, 4, 3, 5, 7, 1}
Output : 3
Groups formed will be (1, 11), (4, 5) and
(3, 7), the difference between highest
sum group (1, 11) i.e 12 and lowest sum
group (4, 5) i.e 9 is 3.
```

Simple Approach: A simple approach would be to try against all combinations of array elements and check against each set of combination difference between the group with the highest sum and the one with lowest sum. A total of n*(n-1)/2 such groups would be formed (nC2).

Time Complexity : O(n^3) To generate groups n^2 iterations will be needed and to check against each group n iterations will be needed and hence n^3 iterations will be needed in worst case.

Efficient Approach: Efficient approach would be to use the greedy approach. Sort the whole array and generate groups by selecting one element from the start of the array and one from the end.

```
 // CPP program to find minimum difference
// between groups of highest and lowest
// sums.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll calculate(ll a[], ll n)
{
    // Sorting the whole array.
    sort(a, a + n);

    // Generating sum groups.
    vector<ll> s;
    for (int i = 0, j = n - 1; i < j; i++, j--)
       s.push_back(a[i] + a[j]);

    ll mini = *min_element(s.begin(), s.end());
    ll maxi = *max_element(s.begin(), s.end());

    return abs(maxi - mini);
}

int main()
{
    ll a[] = { 2, 6, 4, 3 };
    int n = sizeof(a) / (sizeof(a[0]));
    cout << calculate(a, n) << endl;
    return 0;
}
```

Output:

```
1
```

Time Complexity: O (n * log n)

Asked in: Inmobi

Reference: https://www.hackerearth.com/problem/algorithm/project-team/

Source

https://www.geeksforgeeks.org/minimum-difference-between-groups-of-size-two/

Chapter 87

Minimum edges to reverse to make path from a source to a destination

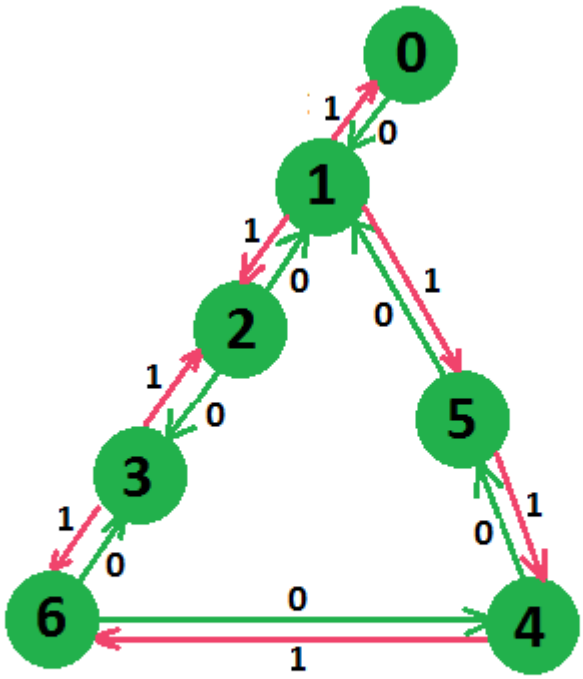
Minimum edges to reverse to make path from a source to a destination - GeeksforGeeks

Given a directed graph and a source node and destination node, we need to find how many edges we need to reverse in order to make at least 1 path from source node to destination node.

Examples:

```
In above graph there were two paths from node 0 to node 6,
0 -> 1 -> 2 -> 3 -> 6
0 -> 1 -> 5 -> 4 -> 6
But for first path only two edges need to be reversed, so answer will be 2 only.
```

This problem can be solved assuming a different version of the given graph. In this version we make a reverse edge corresponding to every edge and we assign that a weight 1 and assign a weight 0 to original edge. After this modification above graph looks something like below,



Modified graph

Now we can see that we have modified the graph in such a way that, if we move towards original edge, no cost is incurred, but if we move toward reverse edge 1 cost is added. So if we apply Dijkstra's shortest path on this modified graph from given source, then that will give us minimum cost to reach from source to destination i.e. minimum edge reversal from source to destination.
Below is the code based on above concept.

```
 // Program to find minimum edge reversal to get
// atleast one path from source to destination
#include <bits/stdc++.h>
using namespace std;
# define INF 0x3f3f3f3f

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V;    // No. of vertices
```

```
    // In a weighted graph, we need to store vertex
    // and weight pair for every edge
    list< pair<int, int> > *adj;

public:
    Graph(int V);  // Constructor

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

    // returns shortest path from s
    vector<int> shortestPath(int s);
};

// Allocates memory for adjacency list
Graph::Graph(int V)
{
    this->V = V;
    adj = new list< pair<int, int> >[V];
}

//  method adds a directed edge from u to v with weight w
void Graph::addEdge(int u, int v, int w)
{
    adj[u].push_back(make_pair(v, w));
}

// Prints shortest paths from src to all other vertices
vector<int> Graph::shortestPath(int src)
{
    // Create a set to store vertices that are being
    // prerocessed
    set< pair<int, int> > setds;

    // Create a vector for distances and initialize all
    // distances as infinite (INF)
    vector<int> dist(V, INF);

    // Insert source itself in Set and initialize its
    // distance as 0.
    setds.insert(make_pair(0, src));
    dist[src] = 0;

    /* Looping till all shortest distance are finalized
       then setds will become empty */
    while (!setds.empty())
    {
```

```
        // The first vertex in Set is the minimum distance
        // vertex, extract it from set.
        pair<int, int> tmp = *(setds.begin());
        setds.erase(setds.begin());

        // vertex label is stored in second of pair (it
        // has to be done this way to keep the vertices
        // sorted distance (distance must be first item
        // in pair)
        int u = tmp.second;

        // 'i' is used to get all adjacent vertices of a vertex
        list< pair<int, int> >::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
        {
            // Get vertex label and weight of current adjacent
            // of u.
            int v = (*i).first;
            int weight = (*i).second;

            //  If there is shorter path to v through u.
            if (dist[v] > dist[u] + weight)
            {
                /*  If distance of v is not INF then it must be in
                    our set, so removing it and inserting again
                    with updated less distance.
                    Note : We extract only those vertices from Set
                    for which distance is finalized. So for them,
                    we would never reach here.  */
                if (dist[v] != INF)
                    setds.erase(setds.find(make_pair(dist[v], v)));

                // Updating distance of v
                dist[v] = dist[u] + weight;
                setds.insert(make_pair(dist[v], v));
            }
        }
    }
    return dist;
}

/* method adds reverse edge of each original edge
   in the graph. It gives reverse edge a weight = 1
   and all original edges a weight of 0. Now, the
   length of the shortest path will give us the answer.
   If shortest path is p: it means we used p reverse
   edges in the shortest path. */
Graph modelGraphWithEdgeWeight(int edge[][2], int E, int V)
```

```
{
    Graph g(V);
    for (int i = 0; i < E; i++)
    {
        //  original edge : weight 0
        g.addEdge(edge[i][0], edge[i][1], 0);

        //  reverse edge : weight 1
        g.addEdge(edge[i][1], edge[i][0], 1);
    }
    return g;
}

// Method returns minimum number of edges to be
// reversed to reach from src to dest
int getMinEdgeReversal(int edge[][2], int E, int V,
                       int src, int dest)
{
    //  get modified graph with edge weight
    Graph g = modelGraphWithEdgeWeight(edge, E, V);

    //  get shortes path vector
    vector<int> dist = g.shortestPath(src);

    // If distance of destination is still INF,
    // not possible
    if (dist[dest] == INF)
        return -1;
    else
        return dist[dest];
}

//  Driver code to test above method
int main()
{
    int V = 7;
    int edge[][2] = {{0, 1}, {2, 1}, {2, 3}, {5, 1},
                     {4, 5}, {6, 4}, {6, 3}};
    int E = sizeof(edge) / sizeof(edge[0]);

    int minEdgeToReverse =
                  getMinEdgeReversal(edge, E, V, 0, 6);
    if (minEdgeToReverse != -1)
        cout << minEdgeToReverse << endl;
    else
        cout << "Not possible" << endl;
    return 0
}
```

Output:

2

Source

https://www.geeksforgeeks.org/minimum-edges-reverse-make-path-source-destination/

Chapter 88

Minimum increment by k operations to make all elements equal

Minimum increment by k operations to make all elements equal - GeeksforGeeks

You are given an array of n-elements, you have to find the number of operations needed to make all elements of array equal. Where a single operation can increment an element by k. If it is not possible to make all elements equal print -1.

Example :

```
Input : arr[] = {4, 7, 19, 16},  k = 3
Output : 8

Input : arr[] = {4, 4, 4, 4}, k = 3
Output : 0

Input : arr[] = {4, 2, 6, 8}, k = 3
Output : -1
```

To solve this question we require to check whether all elements can became equal or not and that too only by incrementing k from elements value. For this we have to check that the difference of any two elements should always be divisible by k. If it is so, then all elements can become equal otherwise they can not became equal in any case by incrementing k from them. Also, the number of operations required can be calculated by finding value of (max – Ai)/k for all elements. where max is maximum element of array.
Algorithm :

```
// iterate for all elements
for (int i=0; i<n; i++)
{
    // check if element can make equal to max
    //  or not if not then return -1
    if ((max - arr[i]) % k != 0 )
        return -1;

    // else update res for required operations
    else
        res += (max - arr[i]) / k ;
}

return res;
```

C++

```
 // Program to make all array equal
#include <bits/stdc++.h>
using namespace std;

// function for calculating min operations
int minOps(int arr[], int n, int k)
{
    // max elements of array
    int max = *max_element(arr, arr + n);
    int res = 0;

    // iterate for all elements
    for (int i = 0; i < n; i++) {

        // check if element can make equal to
        // max or not if not then return -1
        if ((max - arr[i]) % k != 0)
            return -1;

        // else update res for required operations
        else
            res += (max - arr[i]) / k;
    }

    // return result
    return res;
}

// driver program
int main()
{
```

```
    int arr[] = { 21, 33, 9, 45, 63 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 6;
    cout << minOps(arr, n, k);
    return 0;
}
```

Java

```
 // Program to make all array equal
import java.io.*;
import java.util.Arrays;

class GFG {
    // function for calculating min operations
    static int minOps(int arr[], int n, int k)
    {
        // max elements of array
        Arrays.sort(arr);
        int max = arr[arr.length - 1];
        int res = 0;

        // iterate for all elements
        for (int i = 0; i < n; i++) {

            // check if element can make equal to
            // max or not if not then return -1
            if ((max - arr[i]) % k != 0)
                return -1;

            // else update res for required operations
            else
                res += (max - arr[i]) / k;
        }

        // return result
        return res;
    }

    // Driver program
    public static void main(String[] args)
    {
        int arr[] = { 21, 33, 9, 45, 63 };
        int n = arr.length;
        int k = 6;
        System.out.println(minOps(arr, n, k));
    }
}
```

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

Python3

```
 # Python3 Program to make all array equal

# function for calculating min operations
def minOps(arr, n, k):

    # max elements of array
    max1 = max(arr)
    res = 0

    # iterate for all elements
    for i in range(0, n):

        # check if element can make equal to
        # max or not if not then return -1
        if ((max1 - arr[i]) % k != 0):
            return -1

        # else update res fo
        # required operations
        else:
            res += (max1 - arr[i]) / k

    # return result
    return int(res)

# driver program
arr = [21, 33, 9, 45, 63]
n = len(arr)
k = 6
print(minOps(arr, n, k))

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

C#

```
 // C# program Minimum increment by
// k operations to make all elements equal.
using System;

class GFG {
```

```
    // function for calculating min operations
    static int minOps(int[] arr, int n, int k)
    {

        // max elements of array
        Array.Sort(arr);
        int max = arr[arr.Length - 1];
        int res = 0;

        // iterate for all elements
        for (int i = 0; i < n; i++) {

            // check if element can make
            // equal to max or not if not
            // then return -1
            if ((max - arr[i]) % k != 0)
                return -1;

            // else update res for required
            // operations
            else
                res += (max - arr[i]) / k;
        }

        // return result
        return res;
    }

    // Driver program
    public static void Main()
    {
        int[] arr = { 21, 33, 9, 45, 63 };
        int n = arr.Length;
        int k = 6;

        Console.Write(minOps(arr, n, k));
    }
}

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

PHP

```
 <?php
// Program to make all array equal

// function for calculating
// min operations
```

```
function minOps($arr, $n, $k)
{
    // max elements of array
    $max = max($arr);
    $res = 0;

    // iterate for all elements
    for ($i = 0; $i < $n; $i++)
    {

        // check if element can
        // make equal to max or
        // not if not then return -1
        if (($max - $arr[$i]) % $k != 0)
            return -1;

        // else update res for
        // required operations
        else
            $res += ($max -
                     $arr[$i]) / $k;
    }

    // return result
    return $res;
}

// Driver Code
$arr = array(21, 33, 9, 45, 63);
$n = count($arr);
$k = 6;
print (minOps($arr, $n, $k));

// This code is contributed
// by Manish Shaw(manishshaw1)
?>
```

Output :

```
24
```

Improved By : nitin mittal, manishshaw1

Source

https://www.geeksforgeeks.org/minimum-increment-k-operations-make-elements-equal/

Chapter 89

Minimum increment/decrement to make array non-Increasing

Minimum increment/decrement to make array non-Increasing - GeeksforGeeks

Given an array a, your task is to convert it into a non-increasing form such that we can either increment or decrement the array value by 1 in minimum changes possible.

Examples :

```
Input : a[] = {3, 1, 2, 1}
Output : 1
Explanation :
We can convert the array into 3 1 1 1 by
changing 3rd element of array i.e. 2
into its previous integer 1 in one step
hence only one step is required.

Input : a[] = {3, 1, 5, 1}
Output : 4
We need to decrease 5 to 1 to make array sorted
in non-increasing order.

Input : a[] = {1, 5, 5, 5}
Output : 4
We need to increase 1 to 5.
```

Brute-Force approach : We consider both possibilities for every element and find the minimum of two possibilities.

Efficient Approach : Calculate the sum of absolute differences between the final array elements and the current array elements. Thus, the answer will be the sum of the difference

between the ith element and the smallest element occurred till then. For this, we can maintain a min-heap to find the smallest element encountered till now. In the min-priority queue, we will put the elements and new elements are compared with the previous minimum. If new minimum is found we will update it, this is done because each of the next element which is coming should be smaller than the current minimum element found till. Here, we calculate the difference so that we can get how much we have to change the current number so that it will be equal or less than previous numbers encountered till. Lastly, the sum of all these difference will be our answer as this will give the final value upto which we have to change the elements.

Below is **C++** implementation of the above approach

```
 // CPP code to count the change required to
// convert the array into non-increasing array
#include <bits/stdc++.h>
using namespace std;

int DecreasingArray(int a[], int n)
{
    int sum = 0, dif = 0;

    // min heap
    priority_queue<int, vector<int>, greater<int> > pq;

    // Here in the loop we will
    // check that whether the upcoming
    // element of array is less than top
    // of priority queue. If yes then we
    // calculate the difference. After
    // that we will remove that element
    // and push the current element in
    // queue. And the sum is incremented
    // by the value of difference
    for (int i = 0; i < n; i++) {
        if (!pq.empty() && pq.top() < a[i]) {
            dif = a[i] - pq.top();
            sum += dif;
            pq.pop();
            pq.push(a[i]);
        }
        pq.push(a[i]);
    }

    return sum;
}

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

```
    int a[] = { 3, 1, 2, 1 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << DecreasingArray(a, n);

    return 0;
}
```

Output:

```
1
```

Time Complexity: **O(n log(n))**
Space Complexity: **O(n)**

Also see : Convert to strictly increasing array with minimum changes.

Source

https://www.geeksforgeeks.org/minimum-incrementdecrement-to-make-array-non-increasing/

Chapter 90

Minimum initial vertices to traverse whole matrix with given conditions

Minimum initial vertices to traverse whole matrix with given conditions - GeeksforGeeks

We are given a matrix that contains different values in its each cell. Our aim is to find the minimal set of positions in the matrix such that entire matrix can be traversed starting from the positions in the set.
We can traverse the matrix under below conditions:

- We can move only to those neighbors that contain value less than or to equal to the current cell's value. A neighbor of cell is defined as the cell that shares a side with the given cell.

Examples:

```
Input : 1 2 3
        2 3 1
        1 1 1
Output : 1 1
         0 2
If we start from 1, 1 we can cover 6
vertices in the order 1, 1 -> 1, 0 -> 2, 0
-> 2, 1 -> 2, 2 -> 1, 2. We cannot cover
the entire matrix with this vertex. Remaining
vertices can be covered 0, 2 -> 0, 1 -> 0, 0.

Input : 3 3
        1 1
```

```
Output : 0 1
If we start from 0, 1, we can traverse
the entire matrix from this single vertex
in this order 0, 0 -> 0, 1 -> 1, 1 -> 1, 0.
Traversing the matrix in this order
satisfies all the conditions stated above.
```

From the above examples, we can easily identify that in order to use minimum number of positions we have to start from the positions having highest cell value. Therefore we pick the positions that contain the highest value in the matrix. We take the vertices having highest value in separate array. We perform DFS on every vertex starting from the highest value. If we encounter any unvisited vertex during dfs then we have to include this vertex in our set. When all the cells have been processed then the set contains the required vertices.

How does this work?
We need to visit all vertices and to reach largest values we must start with them. If two largest values are not adjacent, then both of them must be picked. If two largest values are adjacent, then any of them can be picked as moving to equal value neighbors is allowed.

```
 // CPP program to find minimum initial
// vertices to reach whole matrix.
#include <bits/stdc++.h>
using namespace std;

const int MAX = 100;

// (n, m) is current source cell from which
// we need to do DFS. N and M are total no.
// of rows and columns.
void dfs(int n, int m, bool visit[][MAX],
         int adj[][MAX], int N, int M)
{
    // Marking the vertex as visited
    visit[n][m] = 1;

    // If below neighbor is valid and has
    // value less than or equal to current
    // cell's value
    if (n + 1 < N &&
        adj[n][m] >= adj[n + 1][m] &&
        !visit[n + 1][m])
        dfs(n + 1, m, visit, adj, N, M);

    // If right neighbor is valid and has
    // value less than or equal to current
    // cell's value
    if (m + 1 < M &&
        adj[n][m] >= adj[n][m + 1] &&
```

```
        !visit[n][m + 1])
        dfs(n, m + 1, visit, adj, N, M);

    // If above neighbor is valid and has
    // value less than or equal to current
    // cell's value
    if (n - 1 >= 0 &&
        adj[n][m] >= adj[n - 1][m] &&
        !visit[n - 1][m])
        dfs(n - 1, m, visit, adj, N, M);

    // If left neighbor is valid and has
    // value less than or equal to current
    // cell's value
    if (m - 1 >= 0 &&
        adj[n][m] >= adj[n][m - 1] &&
        !visit[n][m - 1])
        dfs(n, m - 1, visit, adj, N, M);
}

void printMinSources(int adj[][MAX], int N, int M)
{
    // Storing the cell value and cell indices
    // in a vector.
    vector<pair<long int, pair<int, int> > > x;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < M; j++)
            x.push_back(make_pair(adj[i][j],
                        make_pair(i, j)));


    // Sorting the newly created array according
    // to cell values
    sort(x.begin(), x.end());

    // Create a visited array for DFS and
    // initialize it as false.
    bool visit[N][MAX];
    memset(visit, false, sizeof(visit));

    // Applying dfs for each vertex with
    // highest value
    for (int i = x.size()-1; i >=0 ; i--)
    {
        // If the given vertex is not visited
        // then include it in the set
        if (!visit[x[i].second.first][x[i].second.second])
        {
```

```
            cout << x[i].second.first << " "
                 << x[i].second.second << endl;
            dfs(x[i].second.first, x[i].second.second,
               visit, adj, N, M);
        }
    }
}

// Driver code
int main()
{
    int N = 2, M = 2;

    int adj[N][MAX] = {{3, 3},
                       {1, 1}};
    printMinSources(adj, N, M);
    return 0;
}
```

Output:

```
0 1
```

Source

https://www.geeksforgeeks.org/minimum-initial-vertices-traverse-whole-matrix-given-conditions/

Chapter 91

Minimum number of adjacent swaps for arranging similar elements together

Minimum number of adjacent swaps for arranging similar elements together - GeeksforGeeks

Given an array of 2 * N positive integers where each array element lies between 1 to N and appears exactly twice in the array. The task is to find the minimum number of adjacent swaps required to arrange all similar array elements together.

Note: It is not necessary that the final array (after performing swaps) should be sorted.

Examples:

> Input: arr[] = { 1, 2, 3, 3, 1, 2 }
> Output: 5
> After first swapping, array will be arr[] = { 1, 2, 3, 1, 3, 2 },
> after second arr[] = { 1, 2, 1, 3, 3, 2 }, after third arr[] = { 1, 1, 2, 3, 3, 2 },
> after fourth arr[] = { 1, 1, 2, 3, 2, 3 }, after fifth arr[] = { 1, 1, 2, 2, 3, 3 }
>
> Input: arr[] = { 1, 2, 1, 2 }
> Output: 1
> arr[2] can be swapped with arr[1] to get the required position.

Approach : This problem can be solved using greedy approach. Following are the steps :

1. Keep an array *visited[]* which tells that visited[curr_ele] is false if swap operation has not been performed on curr_ele.
2. Traverse through the original array and if the current array element has not been visited yet i.e. *visited[arr[curr_ele]] = false*, set it to true and iterate over another loop starting from the current position to the end of array.

3. Initialize a variable count which will determine the number of swaps required to place the current element's partner at its correct position.
4. In nested loop, increment count only if the visited[curr_ele] is false (since if it is true, means curr_ele has already been placed at its correct position).
5. If the current element's partner is found in the nested loop, add up the value of count to the total answer.

Below is the implementation of above approach:

```
 // C++ Program to find the minimum number of
// adjacent swaps to arrange similar items together

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

// Function to find minimum swaps
int findMinimumAdjacentSwaps(int arr[], int N)
{
    // visited array to check if value is seen already
    bool visited[N + 1];

    int minimumSwaps = 0;
    memset(visited, false, sizeof(visited));

    for (int i = 0; i < 2 * N; i++) {

        // If the arr[i] is seen first time
        if (visited[arr[i]] == false) {
            visited[arr[i]] = true;

            // stores the number of swaps required to
            // find the correct position of current
            // element's partner
            int count = 0;

            for (int j = i + 1; j < 2 * N; j++) {

                // Increment count only if the current
                // element has not been visited yet (if is
                // visited, means it has already been placed
                // at its correct position)
                if (visited[arr[j]] == false)
                    count++;

                // If current element's partner is found
                else if (arr[i] == arr[j])
```

```
                    minimumSwaps += count;
                }
            }
        }
        return minimumSwaps;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 3, 1, 2 };
    int N = sizeof(arr) / sizeof(arr[0]);
    N /= 2;

    cout << findMinimumAdjacentSwaps(arr, N) << endl;
    return 0;
}
```

Output:

```
5
```

Time Complexity: $O(N^2)$

Source

https://www.geeksforgeeks.org/minimum-number-of-adjacent-swaps-for-arranging-similar-elements-together/

Chapter 92

Minimum number of days required to complete the work

Minimum number of days required to complete the work - GeeksforGeeks

Given N works numbered from 1 to N. Given two arrays D1[] and D2[] of N elements each. Also, each work number W(i) is assigned days, D1[i] and D2[i](*Such that, D2[i] < D1[i]*) either on which it can be completed.

Also, it is mentioned that each work has to completed according to **non-decreasing date of the array D1[]**.

The task is to find the **minimum number of days** required to complete the work in a non-decreasing order of days in D1[].

Examples:

Input :
N = 3
D1[] = {5, 3, 4}
D2[] = {2, 1, 2}
Output : 2

Explanation:
3 works are to be completed. The first value on Line(i) is D1(i) and the second value is D2(i) where D2(i) < D1(i). The smart worker can finish the second work on Day 1 and then both third work and first work in Day 2, thus maintaining the non-decreasing order of D1[], [3 4 5].

Input :
N = 6
D1[] = {3, 3, 4, 4, 5, 5}
D2[] = {1, 2, 1, 2, 4, 4}
Output : 4

Approach: The solution is greedy. The work(i) can be sorted by increasing D1[i], breaking the ties by increasing D2[i]. If we consider the works in this order, we can try to finish the works as early as possible. First of all complete the first work on D2[1]. Move to the second work. If we can complete it on day D2[2] such that (D2[1]<=D2[2]), do it. Otherwise, do the work on day D[2]. Repeat the process until we complete the N-th work, keeping the day of the latest work.

Below is the implementation of the above approach
C++

```
 // C++ program to find the minimum
// number days required

#include <bits/stdc++.h>
using namespace std;
#define inf INT_MAX

// Function to find the minimum
// number days required
int minimumDays(int N, int D1[], int D2[])
{
    // initialising ans to least value possible
    int ans = -inf;

    // vector to store the pair of D1(i) and D2(i)
    vector<pair<int, int> > vect;

    for (int i = 0; i < N; i++)
        vect.push_back(make_pair(D1[i], D2[i]));


    // sort by first i.e D(i)
    sort(vect.begin(), vect.end());

    // Calculate the minimum possible days
    for (int i = 0; i < N; i++) {
        if (vect[i].second >= ans)
            ans = vect[i].second;
        else
            ans = vect[i].first;
    }

    // return the answer
    return ans;
}

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

```
    // Number of works
    int N = 3;

    // D1[i]
    int D1[] = { 6, 5, 4 };

    // D2[i]
    int D2[] = { 1, 2, 3 };

    cout<<minimumDays(N, D1, D2);

    return 0;
}
```

Java

```
// Java program to find the minimum
// number days required
import java.util.*;
import java.lang.*;
import java.io.*;

// pair class for number of days
class Pair
{
int x, y;

Pair(int a, int b)
{
this.x = a;
this.y = b;
}
}

class GFG
{
static int inf = Integer.MIN_VALUE;

// Function to find the minimum
// number days required
public static int minimumDays(int N, int D1[],
int D2[])
{
// initialising ans to
// least value possible
int ans = -inf;

ArrayList
list = new ArrayList();

for (int i = 0; i < N; i++) list.add(new Pair(D1[i], D2[i])); // sort by first i.e D(i) Collections.sort(list, new Comparator()
```

```
{
@Override
public int compare(Pair p1, Pair p2)
{
return p1.x – p2.x;
}
});

// Calculate the minimum possible days
for (int i = 0; i < N; i++) { if (list.get(i).y >= ans)
ans = list.get(i).y;
else
ans = list.get(i).x;
}

return ans;
}

// Driver Code
public static void main (String[] args)
{
// Number of works
int N = 3;

// D1[i]
int D1[] = new int[]{6, 5, 4};

// D2[i]
int D2[] = new int[]{1, 2, 3};

System.out.print(minimumDays(N, D1, D2));
}
}

// This code is contributed by Kirti_Mangal
```

Output:

```
6
```

Improved By : Kirti_Mangal

Source

https://www.geeksforgeeks.org/minimum-number-of-days-required-to-complete-the-work/

Chapter 93

Minimum number of operations to convert a given sequence into a Geometric Progression

Minimum number of operations to convert a given sequence into a Geometric Progression - GeeksforGeeks

Given a sequence of N elements, only three operations can be performed on any element at most one time. The operations are:

1. Add one to the element.
2. Subtract one from the element.
3. Leave the element unchanged.

Perform any one of the operations on all elements in the array. The task is to find the minimum number of operations(addition and subtraction) that can be performed on the sequence, in order to convert it into a Geometric Progression. If it is not possible to generate a GP by performing the above operations, print -1.

Examples:

> **Input**: a[] = {1, 1, 4, 7, 15, 33}
> **Output**: The minimum number of operations are 4.
> Steps:
>
> 1. Keep a_1 unchanged
> 2. Add one to a_2.
> 3. Keep a_3 unchanged
> 4. Subtract one from a_4.

5. Subtract one from a_5.
6. Add one to a_6.

The resultant sequence is {1, 2, 4, 8, 16, 32}

Input: a[] = {20, 15, 20, 15}
Output: -1

Approach The key observation to be made here is that any Geometric Progression is uniquely determined by only its first two elements (Since the ratio between each of the next pairs has to be the same as the ratio between this pair, consisting of the first two elements). Since only **3*3** permutations are possible. The possible combination of operations are (+1, +1), (+1, 0), (+1, -1), (-1, +1), (-1, 0), (-1, -1), (0, +1), (0, 0) and (0, -1). Using brute force all these **9** permutations and checking if they form a GP in linear time will give us the answer. The minimum of the operations which result in combinations which are in GP will be the answer.

Below is the implementation of the above approach:

```
 // C++ program to find minimum number
// of operations to convert a given
// sequence to an Geometric Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print the GP series
void construct(int n, pair<double, double> ans_pair)
{
    // Check for possibility
    if (ans_pair.first == -1) {
        cout << "Not possible";
        return;
    }
    double a1 = ans_pair.first;
    double a2 = ans_pair.second;
    double r = a2 / a1;

    cout << "The resultant sequence is:\n";
    for (int i = 1; i <= n; i++) {
        double ai = a1 * pow(r, i - 1);
        cout << ai << " ";
    }
}

// Function for getting the Arithmetic Progression
void findMinimumOperations(double* a, int n)
{
    int ans = INT_MAX;
    // The array c describes all the given set of
```

```
// possible operations.
int c[] = { -1, 0, 1 };
// Size of c
int possiblities = 3;

// candidate answer
int pos1 = -1, pos2 = -1;

// loop through all the permutations of the first two
// elements.
for (int i = 0; i < possiblities; i++) {
    for (int j = 0; j < possiblities; j++) {

        // a1 and a2 are the candidate first two elements
        // of the possible GP.
        double a1 = a[1] + c[i];
        double a2 = a[2] + c[j];

        // temp stores the current answer, including the
        // modification of the first two elements.
        int temp = abs(a1 - a[1]) + abs(a2 - a[2]);

        if (a1 == 0 || a2 == 0)
            continue;

        // common ratio of the possible GP
        double r = a2 / a1;

        // To check if the chosen set is valid, and id yes
        // find the number of operations it takes.
        for (int pos = 3; pos <= n; pos++) {

            // ai is value of a[i] according to the assumed
            // first two elements a1, a2
            // ith element of an GP = a1*((a2-a1)^(i-1))
            double ai = a1 * pow(r, pos - 1);

            // Check for the "proposed" element to be only
            // differing by one
            if (a[pos] == ai) {
                continue;
            }
            else if (a[pos] + 1 == ai || a[pos] - 1 == ai) {
                temp++;
            }
            else {
                temp = INT_MAX; // set the temporary ans
                break; // to infinity and break
```

```
                }
            }

            // update answer
            if (temp < ans) {
                ans = temp;
                pos1 = a1;
                pos2 = a2;
            }
        }
    }
    if (ans == -1) {
        cout << "-1";
        return;
    }

    cout << "Minimum Number of Operations are " << ans << "\n";
    pair<double, double> ans_pair = { pos1, pos2 };

    // Calling function to print the sequence
    construct(n, ans_pair);
}

// Driver Code
int main()
{

    // array is 1-indexed, with a[0] = 0
    // for the sake of simplicity
    double a[] = { 0, 7, 20, 49, 125 };

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

    // Function to print the minimum operations
    // and the sequence of elements
    findMinimumOperations(a, n - 1);
    return 0;
}
```

Output:

```
Minimum Number of Operations are 2
The resultant sequence is:
8 20 50 125
```

Time Complexity : O(9*N)

Source

https://www.geeksforgeeks.org/minimum-number-of-operations-to-convert-a-given-sequence-into-a-geometric-prog

Chapter 94

Minimum operations to make GCD of array a multiple of k

Minimum operations to make GCD of array a multiple of k - GeeksforGeeks

Given an array and k, we need to find the minimum operations needed to make GCD of the array equal or multiple of k. Here an operation mean either increment or decrement an array element by 1.

Examples:

```
Input : a = { 4, 5, 6 }, k = 5
Output : 2
Explanation : We can increase 4 by 1 so that it becomes 5 and decrease 6 by 1 so that it becomes

Input : a = { 4, 9, 6 }, k = 5
Output : 3
Explanation :Just like previous example we can increase and decrease 4 and 6 by 1 and increase 9
```

Here we have to make the gcd of the array equal or multiple to k, which means there will be cases in which all the elements are near are greater than k or is near to some of its multiple. So, to solve this we just have to make each array value equal to or multiple to K. By doing this we will achieve our solution, as if every element is multiple of k then it's GCD will be atleast K. Now our next target is to convert the array elements in the minimum operation i.e. minimum number of increment and decrement. This minimum value of increment or decrement can be known only by taking the remainder of each number from K i.e. *either we have to take the remainder value or (k-remainder) value, whichever is minimum among them.*

Below is implementation of this approach

C++

```
// CPP program to make GCD of array a mutiple
```

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

int MinOperation(int a[], int n, int k)
{

    int result = 0;

    for (int i = 0; i < n; ++i) {

        // If array value is not 1
        // and it is greater than k
        // then we can increase the
        // or decrease the remainder
        // obtained by dividing k
        // from the ith value of array
        // so that we get the number
        // which is either closer to k
        // or its multiple
        if (a[i] != 1 && a[i] > k) {
            result = result + min(a[i] % k, k - a[i] % k);
        }
        else {

            // Else we only have one choice
            // which is to increment the value
            // to make equal to k
            result = result + k - a[i];
        }
    }

    return result;
}

// Driver code
int main()
{
    int arr[] = { 4, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 5;
    cout << MinOperation(arr, n, k);
    return 0;
}
```

Java

```
 // Java program to make GCD
```

```
// of array a mutiple of k.
import java.io.*;

class GFG
{
static int MinOperation(int a[],
                        int n, int k)
{

    int result = 0;

    for (int i = 0; i < n; ++i)
    {

        // If array value is not 1
        // and it is greater than k
        // then we can increase the
        // or decrease the remainder
        // obtained by dividing k
        // from the ith value of array
        // so that we get the number
        // which is either closer to k
        // or its multiple
        if (a[i] != 1 && a[i] > k)
        {
            result = result +
                     Math.min(a[i] % k,
                          k - a[i] % k);
        }
        else
        {

            // Else we only have one
            // choice which is to
            // increment the value
            // to make equal to k
            result = result + k - a[i];
        }
    }

    return result;
}

// Driver code
public static void main (String[] args)
{
    int arr[] = {4, 5, 6};
    int n = arr.length;
```

```
    int k = 5;
    System.out.println(MinOperation(arr, n, k));
}
}

// This code is contributed
// by akt_mit
```

PHP

```
 <?php
// PHP program to make
// GCD of array a mutiple
// of k.

function MinOperation($a, $n, $k)
{
    $result = 0;

    for ($i = 0; $i < $n; ++$i)
    {

        // If array value is
        // not 1 and it is
        // greater than k then
        // we can increase the
        // or decrease the remainder
        // obtained by dividing
        // k from the ith value of
        // array so that we get the
        // number which is either
        // closer to k or its multiple
        if ($a[$i] != 1 && $a[$i] > $k)
        {
            $result = $result + min($a[$i] %
                                    $k, $k -
                                    $a[$i] % $k);
        }
        else
        {

            // Else we only have one
            // choice which is to
            // increment the value
            // to make equal to k
            $result = $result +
                      $k - $a[$i];
        }
```

```
    }

    return $result;
}

// Driver code
$arr = array(4, 5, 6);
$n = sizeof($arr) /
     sizeof($arr[0]);
$k = 5;
echo MinOperation($arr, $n, $k);

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

Output:

```
2
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/minimum-operations-make-gcd-array-multiple-k/

Chapter 95

Minimum product subset of an array

Minimum product subset of an array - GeeksforGeeks

Given an array a, we have to find minimum product possible with the subset of elements present in the array. The minimum product can be single element also.

Examples:

```
Input : a[] = { -1, -1, -2, 4, 3 }
Output : -24
Explanation : Minimum product will be ( -2 * -1 * -1 * 4 * 3 ) = -24

Input : a[] = { -1, 0 }
Output : -1
Explanation : -1(single element) is minimum product possible

Input : a[] = { 0, 0, 0 }
Output : 0
```

A simple solution is to generate all subsets, find product of every subset and return maximum product.

A better solution is to use the below facts.

1. If there are even number of negative numbers and no zeros, the result is the product of all except the largest valued negative number.
2. If there are an odd number of negative numbers and no zeros, the result is simply the product of all.

3. If there are zeros and positive, no negative, the result is 0. The exceptional case is when there is no negative number and all other elements positive then our result should be the first minimum positive number.

C++

```
 // CPP program to find maximum product of
// a subset.
#include <bits/stdc++.h>
using namespace std;

int minProductSubset(int a[], int n)
{
    if (n == 1)
        return a[0];

    // Find count of negative numbers, count
    // of zeros, maximum valued negative number,
    // minimum valued positive number and product
    // of non-zero numbers
    int max_neg = INT_MIN;
    int min_pos = INT_MAX;
    int count_neg = 0, count_zero = 0;
    int prod = 1;
    for (int i = 0; i < n; i++) {

        // If number is 0, we don't
        // multiply it with product.
        if (a[i] == 0) {
            count_zero++;
            continue;
        }

        // Count negatives and keep
        // track of maximum valued negative.
        if (a[i] < 0) {
            count_neg++;
            max_neg = max(max_neg, a[i]);
        }

        // Track minimum positive
        // number of array
        if (a[i] > 0)
            min_pos = min(min_pos, a[i]);

        prod = prod * a[i];
    }
```

```
    // If there are all zeros
    // or no negative number present
    if (count_zero == n ||
       (count_neg == 0 && count_zero > 0))
        return 0;

    // If there are all positive
    if (count_neg == 0)
        return min_pos;

    // If there are even number of
    // negative numbers and count_neg not 0
    if (!(count_neg & 1) && count_neg != 0) {

        // Otherwise result is product of
        // all non-zeros divided by maximum
        // valued negative.
        prod = prod / max_neg;
    }

    return prod;
}

int main()
{
    int a[] = { -1, -1, -2, 4, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << minProductSubset(a, n);
    return 0;
}
```

Java

```
 // Java program to find maximum product of
// a subset.
class GFG {

    static int minProductSubset(int a[], int n)
    {
        if (n == 1)
            return a[0];

        // Find count of negative numbers,
        // count of zeros, maximum valued
        // negative number, minimum valued
        // positive number and product of
        // non-zero numbers
        int negmax = Integer.MIN_VALUE;
```

```
int posmin = Integer.MIN_VALUE;
int count_neg = 0, count_zero = 0;
int product = 1;

for (int i = 0; i < n; i++)
{

    // if number is zero,count it
    // but dont multiply
    if(a[i] == 0){
        count_zero++;
        continue;
    }

// count the negetive numbers
// and find the max negetive number
if(a[i] < 0)
{
        count_neg++;
        negmax = Math.max(negmax, a[i]);
    }

    // find the minimum positive number
    if(a[i] > 0 && a[i] < posmin)
    posmin = a[i];

    product *= a[i];
}

// if there are all zeroes
// or zero is present but no
// negetive number is present
if (count_zero == n ||
    (count_neg == 0 && count_zero > 0))
    return 0;

// If there are all positive
if (count_neg == 0)
    return posmin;

// If there are even number except
// zero of negative numbers
if (count_neg % 2 == 0 && count_neg != 0)
{

    // Otherwise result is product of
    // all non-zeros divided by maximum
    // valued negative.
```

```
            product = product / negmax;
        }

        return product;
    }

    // main function
    public static void main(String[] args)
    {

        int a[] = { -1, -1, -2, 4, 3 };
        int n = 5;

        System.out.println(minProductSubset(a, n));
    }
}

// This code is contributed by Arnab Kundu.
```

Python3

```
 # Python3 program to find maximum
# product of a subset.

# def to find maximum
# product of a subset
def minProductSubset(a, n) :
    if (n == 1) :
        return a[0]

    # Find count of negative numbers,
    # count of zeros, maximum valued
    # negative number, minimum valued
    # positive number and product
    # of non-zero numbers
    max_neg = float('-inf')
    min_pos = float('inf')
    count_neg = 0
    count_zero = 0
    prod = 1
    for i in range(0,n) :

        # If number is 0, we don't
        # multiply it with product.
        if (a[i] == 0) :
            count_zero = count_zero + 1
            continue
```

```
        # Count negatives and keep
        # track of maximum valued
        # negative.
        if (a[i] < 0) :
            count_neg = count_neg + 1
            max_neg = max(max_neg, a[i])

        # Track minimum positive
        # number of array
        if (a[i] > 0) :
            min_pos = min(min_pos, a[i])

        prod = prod * a[i]


    # If there are all zeros
    # or no negative number
    # present
    if (count_zero == n or (count_neg == 0
                    and count_zero > 0)) :
        return 0;

    # If there are all positive
    if (count_neg == 0) :
        return min_pos

    # If there are even number of
    # negative numbers and count_neg
    # not 0
    if ((count_neg & 1) == 0 and
                        count_neg != 0) :

        # Otherwise result is product of
        # all non-zeros divided by
        # maximum valued negative.
        prod = int(prod / max_neg)

    return prod;

# Driver code
a = [ -1, -1, -2, 4, 3 ]
n = len(a)
print (minProductSubset(a, n))
# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# program to find maximum product of
// a subset.
using System;

public class GFG {

    static int minProductSubset(int[] a, int n)
    {
        if (n == 1)
            return a[0];

        // Find count of negative numbers,
        // count of zeros, maximum valued
        // negative number, minimum valued
        // positive number and product of
        // non-zero numbers
        int negmax = int.MinValue;
        int posmin = int.MinValue;
        int count_neg = 0, count_zero = 0;
        int product = 1;

        for (int i = 0; i < n; i++)
        {

            // if number is zero, count it
            // but dont multiply
            if (a[i] == 0) {
                count_zero++;
                continue;
            }

            // count the negetive numbers
            // and find the max negetive number
            if (a[i] < 0) {
                count_neg++;
                negmax = Math.Max(negmax, a[i]);
            }

            // find the minimum positive number
            if (a[i] > 0 && a[i] < posmin) {
                posmin = a[i];
            }

            product *= a[i];
        }

        // if there are all zeroes
        // or zero is present but no
```

```
        // negetive number is present
        if (count_zero == n || (count_neg == 0
                            && count_zero > 0))
            return 0;

        // If there are all positive
        if (count_neg == 0)
            return posmin;

        // If there are even number except
        // zero of negative numbers
        if (count_neg % 2 == 0 && count_neg != 0)
        {

            // Otherwise result is product of
            // all non-zeros divided by maximum
            // valued negative.
            product = product / negmax;
        }

        return product;
    }

    // main function
    public static void Main()
    {

        int[] a = new int[] { -1, -1, -2, 4, 3 };
        int n = 5;

        Console.WriteLine(minProductSubset(a, n));
    }
}

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

PHP

```
 <?php
// PHP program to find maximum
// product of a subset.

// Function to find maximum
// product of a subset
function minProductSubset($a, $n)
{

    if ($n == 1)
```

```
    return $a[0];

// Find count of negative numbers,
// count of zeros, maximum valued
// negative number, minimum valued
// positive number and product
// of non-zero numbers
$max_neg = PHP_INT_MIN;
$min_pos = PHP_INT_MAX;
$count_neg = 0; $count_zero = 0;
$prod = 1;
for ($i = 0; $i < $n; $i++)
{

    // If number is 0, we don't
    // multiply it with product.
    if ($a[$i] == 0)
    {
        $count_zero++;
        continue;
    }

    // Count negatives and keep
    // track of maximum valued
    // negative.
    if ($a[$i] < 0)
    {
        $count_neg++;
        $max_neg = max($max_neg, $a[$i]);
    }

    // Track minimum positive
    // number of array
    if ($a[$i] > 0)
        $min_pos = min($min_pos, $a[$i]);

    $prod = $prod * $a[$i];
}

// If there are all zeros
// or no negative number
// present
if ($count_zero == $n ||
   ($count_neg == 0 &&
    $count_zero > 0))
    return 0;

// If there are all positive
```

```
    if ($count_neg == 0)
        return $min_pos;

    // If there are even number of
    // negative numbers and count_neg
    // not 0
    if (!($count_neg & 1) &&
          $count_neg != 0)
    {

        // Otherwise result is product of
        // all non-zeros divided by maximum
        // valued negative.
        $prod = $prod / $max_neg;
    }

    return $prod;
}

// Driver code
$a = array( -1, -1, -2, 4, 3 );
$n = sizeof($a);
echo(minProductSubset($a, $n));

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

Output:

```
-24
```

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

Improved By : jit_t, andrew1234, manishshaw1

Source

https://www.geeksforgeeks.org/minimum-product-subset-array/

Chapter 96

Minimum rooms for m events of n batches with given schedule

Minimum rooms for m events of n batches with given schedule - GeeksforGeeks

There are n student groups at the school. On each day in school, there are m time slots. A student group may or may not be free during a time slot. We are given n binary string where each binary string is of length m. A character at j-th position in i-th string is 0 if i-th group is free in j-th slot and 1 if i-th group is busy.

Our task is to determine the minimum number of rooms needed to hold classes for all groups on a single study day. Note that one room can hold at most one group class in a single time slot.

Examples:

> Input : n = 2, m = 7, slots[] = {“0101010”, “1010101”}
> Output : 1
> Explanation : Both group can hold their classes in a single room as they have alternative classes.
>
> Input : n = 3, m = 7, slots[] = {“0101011”, “0011001”, “0110111”}
> Output : 3

Approach used: Here we traverse through each character of strings we have and while traversing maintaining a count of the number of 1’s at each position of the strings and hence we know the number of coinciding classes at each particular time slot. Then we just need to find the maximum number of coinciding classes amongst all time slots.

CPP

```
 // CPP program to find minimum number of rooms
// required
#include <bits/stdc++.h>
```

```
using namespace std;

// Returns minimum number of rooms required
// to perform classes of n groups in m slots
// with given schedule.
int findMinRooms(string slots[], int n, int m)
{
    // Store count of classes happening in
    // every slot.
    int counts[m] = { 0 };
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (slots[i][j] == '1')
                counts[j]++;

    // Number of rooms required is equal to
    // maximum classes happening in a
    // particular slot.
    return *max_element(counts, counts+m);
}

// Driver Code
int main()
{
    int n = 3, m = 7;
    string slots[n] = { "0101011",
                        "0011001",
                        "0110111" };
    cout << findMinRooms(slots, n, m);
    return 0;
}
```

Java

```
 // java program to find the minimum number
// of rooms required
class GFG {

    // Returns minimum number of rooms required
    // to perform classes of n groups in m slots
    // with given schedule.
    static int findMinRooms(String slots[],
                                    int n, int m)
    {

        // Store number of class happening in
        //empty slot
        int counts[] = new int[m];
```

```
        //initilize all values to zero
        for (int i = 0; i < m; i++)
            counts[i] = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (slots[i].charAt(j) == '1')
                    counts[j]++;

        // Number of rooms required is equal to
        // maximum classes happening in a
        // particular slot.

        int max = -1;
        // find the max element
        for (int i = 0; i < m; i++)
            if(max < counts[i])
                max = counts[i];

        return max;
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 3, m = 7;
        String slots[] = { "0101011",
                           "0011001",
                           "0110111" };
        System.out.println( findMinRooms(slots, n, m));
    }
}

// This code is contributed by Arnab Kundu.
```

C#

```
 // C# program to find the minimum number
// of rooms required
using System;
class GFG {

    // Returns minimum number of rooms required
    // to perform classes of n groups in m slots
    // with given schedule.
    static int findMinRooms(string []slots,
                                int n, int m)
```

```
    {

        // Store number of class happening in
        //empty slot
        int []counts = new int[m];

        //initilize all values to zero
        for (int i = 0; i < m; i++)
            counts[i] = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (slots[i][j] == '1')
                    counts[j]++;

        // Number of rooms required is equal to
        // maximum classes happening in a
        // particular slot.

        int max = -1;
        // find the max element
        for (int i = 0; i < m; i++)
            if(max < counts[i])
                max = counts[i];

        return max;
    }

    // Driver Code
    public static void Main()
    {
        int n = 3, m = 7;
        String []slots = { "0101011",
                           "0011001",
                           "0110111" };
        Console.Write( findMinRooms(slots, n, m));
    }
}

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

Output:

```
3
```

Time Complexity : O(m * n)

Auxiliary Space : O(m)

Improved By : andrew1234, nitin mittal

Source

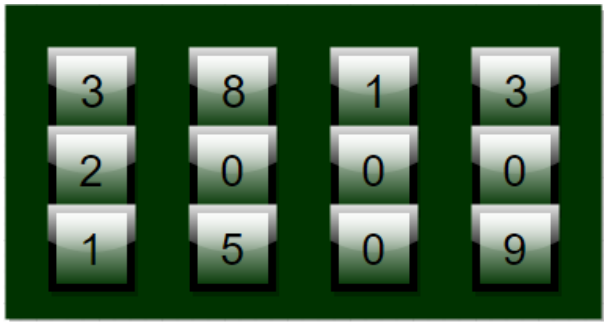
https://www.geeksforgeeks.org/minimum-rooms-for-m-events-of-n-batches-with-given-schedule/

Chapter 97

Minimum rotations to unlock a circular lock

Minimum rotations to unlock a circular lock - GeeksforGeeks

You are given a lock which is made up of n-different circular rings and each ring has 0-9 digit printed serially on it. Initially all n-rings together show a n-digit integer but there is particular code only which can open the lock. You can rotate each ring any number of time in either direction. You have to find the minimum number of rotation done on rings of lock to open the lock.



Examples:

```
Input : Input = 2345, Unlock code = 5432
Output : Rotations required = 8
Explanation : 1st ring is rotated thrice as 2->3->4->5
              2nd ring is rotated once as 3->4
              3rd ring is rotated once as 4->3
```

```
              4th ring is rotated thrice as 5->4->3->2

Input : Input = 1919, Unlock code = 0000
Output : Rotations required = 4
Explanation : 1st ring is rotated once as 1->0
              2nd ring is rotated once as 9->0
              3rd ring is rotated once as 1->0
              4th ring is rotated once as 9->0
```

For a single ring we can rotate it in any of two direction forward or backward as:

- 0->1->2….->9->0
- 9->8->….0->9

But we are concerned with minimum number of rotation required so we should choose *min (abs(a-b), 10-abs(a-b))* as *a-b* denotes the number of forward rotation and *10-abs(a-b)* denotes the number of backward rotation for a ring to rotate from a to b. Further we have to find minimum number for each ring that is for each digit. So starting from right most digit we can easily the find minimum number of rotation required for each ring and end up at left most digit.
C++

```
 // CPP program for min rotation to unlock
#include <bits/stdc++.h>
using namespace std;

// function for min rotation
int minRotation(int input, int unlock_code)
{
    int rotation = 0;
    int input_digit, code_digit;

    // iterate till input and unlock code become 0
    while (input || unlock_code) {

        // input and unlock last digit as reminder
        input_digit = input % 10;
        code_digit = unlock_code % 10;

        // find min rotation
        rotation += min(abs(input_digit - code_digit),
                   10 - abs(input_digit - code_digit));

        // update code and input
        input /= 10;
        unlock_code /= 10;
```

```
    }

    return rotation;
}

// driver code
int main()
{
    int input = 28756;
    int unlock_code = 98234;
    cout << "Minimum Rotation = "
        << minRotation(input, unlock_code);
    return 0;
}
```

Java

```
 // Java program for min rotation to unlock
class GFG
{

    // function for min rotation
    static int minRotation(int input, int unlock_code)
    {
        int rotation = 0;
        int input_digit, code_digit;

        // iterate till input and unlock code become 0
        while (input>0 || unlock_code>0) {

            // input and unlock last digit as reminder
            input_digit = input % 10;
            code_digit = unlock_code % 10;

            // find min rotation
            rotation += Math.min(Math.abs(input_digit
                        - code_digit), 10 - Math.abs(
                           input_digit - code_digit));

            // update code and input
            input /= 10;
            unlock_code /= 10;
        }

        return rotation;
    }

    // driver code
```

```
    public static void main (String[] args) {
    int input = 28756;
    int unlock_code = 98234;
    System.out.println("Minimum Rotation = "+
                  minRotation(input, unlock_code));
    }
}

/* This code is contributed by Mr. Somesh Awasthi */
```

C#

```
 // C# program for min rotation to unlock
using System;

class GFG {

    // function for min rotation
    static int minRotation(int input,
                           int unlock_code)
    {
        int rotation = 0;
        int input_digit, code_digit;

        // iterate till input and
        // unlock code become 0
        while (input > 0 ||
               unlock_code > 0)
        {

            // input and unlock last
            // digit as reminder
            input_digit = input % 10;
            code_digit = unlock_code % 10;

            // find min rotation
            rotation += Math.Min(Math.Abs(input_digit -
                        code_digit), 10 - Math.Abs(
                        input_digit - code_digit));

            // update code and input
            input /= 10;
            unlock_code /= 10;
        }

        return rotation;
    }
```

```
    // Driver Code
    public static void Main ()
    {
        int input = 28756;
        int unlock_code = 98234;
        Console.Write("Minimum Rotation = "+
                        minRotation(input, unlock_code));
    }
}

// This code is contributed by Nitin Mittal
```

PHP

```
 <?php
// PHP program for min
// rotation to unlock


// function for min rotation
function minRotation($input,
                     $unlock_code)
{
    $rotation = 0;
    $input_digit; $code_digit;

    // iterate till input and
    // unlock code become 0
    while ($input || $unlock_code)
    {

        // input and unlock last
        // digit as reminder
        $input_digit = $input % 10;
        $code_digit = $unlock_code % 10;

        // find min rotation
        $rotation += min(abs($input_digit - $code_digit),
                     10 - abs($input_digit - $code_digit));

        // update code and input
        $input /= 10;
        $unlock_code /= 10;
    }

    return $rotation;
}
```

```
    // Driver Code
    $input = 28756;
    $unlock_code = 98234;
    echo "Minimum Rotation = "
        , minRotation($input, $unlock_code);

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

Output:

```
Minimum Rotation = 12
```

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/minimum-rotations-unlock-circular-lock/

Chapter 98

Minimum sum by choosing minimum of pairs from array

Minimum sum by choosing minimum of pairs from array - GeeksforGeeks

Given an array A[] of n-elements. We need to select two adjacent elements and delete the larger of them and store smaller of them to another array say B[]. We need to perform this operation till array A[] contains only single element. Finally, we have to construct the array B[] in such a way that total sum of its element is minimum. Print the total sum of array B[].

Examples:

```
Input : A[] = {3, 4}
Output : 3

Input : A[] = {2, 4, 1, 3}
Output : 3
```

There is an easy trick to solve this question and that is always choose the smallest element of array A[] and its adjacent, delete the adjacent element and copy smallest one to array B[]. Again for next iteration we have same smallest element and any random adjacent element which is to be deleted. After n-1 operations all of elements of A[] got deleted except the smallest one and at the same time array B[] contains "n-1" elements and all are equal to smallest element of array A[].
Thus total sum of array B[] is equal to **smallest * (n-1)**.

C++

```
 // CPP program to minimize the cost
// of array minimization
#include <bits/stdc++.h>
```

```
using namespace std;

// Returns minimum possible sum in
// array B[]
int minSum(int A[], int n)
{
    int min_val = *min_element(A, A+n);
    return (min_val * (n-1));
}

// driver function
int main()
{
    int A[] = { 3, 6, 2, 8, 7, 5};
    int n = sizeof(A)/ sizeof (A[0]);
    cout << minSum(A, n);
    return 0;
}
```

Python

```
 # Python code for minimum cost of
# array minimization

# Function defintion for minCost
def minSum(A):

    # find the minimum element of A[]
    min_val = min(A);

    # return the answer
    return min_val * (len(A)-1)

# driver code
A = [7, 2, 3, 4, 5, 6]
print (minSum(A))
```

C#

```
 // C# program to minimize the
// cost of array minimization
using System;
using System.Linq;

public class GFG
{
```

```
// Returns minimum possible
// sum in array B[]
static int minSum(int []A, int n)
{
    int min_val = A.Min();
    return (min_val * (n - 1));
}

    // Driver Code
    static public void Main()
    {
        int []A = {3, 6, 2, 8, 7, 5};
        int n = A.Length;
        Console.WriteLine(minSum(A, n));

    }
}

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

PHP

```
 <?php
// PHP program to minimize the
// cost of array minimization

// Returns minimum possible
// sum in array B[]
function minSum($A, $n)
{
    $min_val = min($A);
    return ($min_val * ($n - 1));
}

    // Driver Code
    $A = array(3, 6, 2, 8, 7, 5);
    $n = count($A);
    echo minSum($A, $n);

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

Output:

```
10
```

Time Complexity : O(n) in finding the smallest element of the array.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/minimum-sum-choosing-minimum-pairs-array/

Chapter 99

Minimum sum of absolute difference of pairs of two arrays

Minimum sum of absolute difference of pairs of two arrays - GeeksforGeeks

Given two arrays ***a[]*** and ***b[]*** of equal length ***n***. The task is to pair each element of array ***a*** to an element in array ***b***, such that sum ***S*** of **absolute differences of all the pairs is minimum.**

Suppose, two elements **a**[***i***] and **a**[***j***] (***i != j***) of ***a*** are paired with elements **b**[***p***] and **b**[***q***] of ***b*** respectively,
then ***p*** should not be equal to ***q***.

Examples:

```
Input :  a[] = {3, 2, 1}
         b[] = {2, 1, 3}
Output : 0

Input :  n = 4
         a[] = {4, 1, 8, 7}
         b[] = {2, 3, 6, 5}
Output : 6
```

The solution to the problem is a simple greedy approach. It consists of **two** steps.
Step 1 : Sort both the arrays in **O (n log n)** time.
Step 2 : Find absolute difference of each pair of **corresponding elements** *(elements at same index)* of both arrays and add the result to the sum ***S***. The time complexity of this step is **O(n)**.

Hence, the overall time complexity of the program is **O(n log n)**.

C++

```
 // C++ program to find minimum sum of absolute
// differences of two arrays.
#include <bits/stdc++.h>
using namespace std;

// Returns minimum possible pairwise absolute
// difference of two arrays.
long long int findMinSum(int a[], int b[], int n)
{
    // Sort both arrays
    sort(a, a+n);
    sort(b, b+n);

    // Find sum of absolute differences
    long long int sum= 0 ;
    for (int i=0; i<n; i++)
        sum = sum + abs(a[i]-b[i]);

    return sum;
}

// Driver code
int main()
{
    // Both a[] and b[] must be of same size.
    long long int a[] = {4, 1, 8, 7};
    long long int b[] = {2, 3, 6, 5};
    int n = sizeof(a)/sizeof(a[0]);
    printf("%lld\n", findMinSum(a, b, n));
    return 0;
}
```

Java

```
 // Java program to find minimum sum of
// absolute differences of two arrays.
import java.util.Arrays;

class MinSum
{
    // Returns minimum possible pairwise
    // absolute difference of two arrays.
    static long findMinSum(long a[], long b[], long n)
    {
        // Sort both arrays
        Arrays.sort(a);
        Arrays.sort(b);
```

```
        // Find sum of absolute differences
        long sum = 0 ;
        for (int i = 0; i < n; i++)
            sum = sum + Math.abs(a[i] - b[i]);

        return sum;
    }

    // Driver code
    public static void main(String[] args)
    {
        // Both a[] and b[] must be of same size.
        long a[] = {4, 1, 8, 7};
        long b[] = {2, 3, 6, 5};
        int n = a.length;
        System.out.println(findMinSum(a, b, n));
    }
}

// This code is contributed by Raghav Sharma
```

Python3

```
 # Python3 program to find minimum sum
# of absolute differences of two arrays.
def findMinSum(a, b, n):

    # Sort both arrays
    a.sort()
    b.sort()

    # Find sum of absolute differences
    sum = 0

    for i in range(n):
        sum = sum + abs(a[i] - b[i])

    return sum

# Driver program

# Both a[] and b[] must be of same size.
a = [4, 1, 8, 7]
b = [2, 3, 6, 5]
n = len(a)

print(findMinSum(a, b, n))
```

```
# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find minimum sum of
// absolute differences of two arrays.
using System;

class MinSum {

    // Returns minimum possible pairwise
    // absolute difference of two arrays.
    static long findMinSum(long []a, long []b,
                           long n)
    {

        // Sort both arrays
        Array.Sort(a);
        Array.Sort(b);

        // Find sum of absolute differences
        long sum = 0 ;
        for (int i = 0; i < n; i++)
            sum = sum + Math.Abs(a[i] - b[i]);

        return sum;
    }

    // Driver code
    public static void Main(String[] args)
    {
        // Both a[] and b[] must be of same size.
        long []a = {4, 1, 8, 7};
        long []b = {2, 3, 6, 5};
        int n = a.Length;
        Console.Write(findMinSum(a, b, n));
    }
}

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

PHP

```
 <?php
// PHP program to find minimum sum
// of absolute differences of two
// arrays.
```

```
// Returns minimum possible pairwise
// absolute difference of two arrays.
function findMinSum($a, $b, $n)
{

    // Sort both arrays
    sort($a);
    sort($a, $n);
    sort($b);
    sort($b, $n);

    // Find sum of absolute
    // differences
    $sum= 0 ;
    for ($i = 0; $i < $n; $i++)
        $sum = $sum + abs($a[$i] -
                          $b[$i]);

    return $sum;
}

    // Driver Code
    // Both a[] and b[] must
    // be of same size.
    $a = array(4, 1, 8, 7);
    $b = array(2, 3, 6, 5);
    $n = sizeof($a);
    echo(findMinSum($a, $b, $n));

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

Output :

```
6
```

Improved By : parashar, nitin mittal

Source

https://www.geeksforgeeks.org/minimum-sum-absolute-difference-pairs-two-arrays/

Chapter 100

Minimum sum of product of two arrays

Minimum sum of product of two arrays - GeeksforGeeks

Find the minimum sum of Products of two arrays of the same size, given that k modifications are allowed on the first array. In each modification, one array element of the first array can either be increased or decreased by 2.

Examples:

```
Input : ar[] = {1, 2, -3}
        b[]  = {-2, 3, -5}
           k = 5
Output : -31
Explanation:
Here n = 3 and k = 5.
So, we modified a[2], which is -3 and
increased it by 10 (as 5 modifications
are allowed).
Final sum will be :
(1 * -2) + (2 * 3) + (7 * -5)
   -2    +    6    -    35
              -31
(which is the minimum sum of the array
with given conditions)

Input : a[] = {2, 3, 4, 5, 4}
        b[] = {3, 4, 2, 3, 2}
Output : 25
Explanation:
Here, total numbers are 5 and total
```

```
modifications allowed are 3. So, modify
a[1], which is 3 and decreased it by 6
(as 3 modifications are allowed).
Final sum will be :
(2 * 3) + (-3 * 4) + (4 * 2) + (5 * 3) + (4 * 2)
   6    -    12    +    8    +    15   +    8
                      25
(which is the minimum sum of the array with
given conditions)
```

Since we need to minimize the product sum, we find the maximum product and reduce it. By taking some examples, we observe that making 2*k changes to only one element is enough to get the minimum sum. Based on this observation, we consider every element as the element on which we apply all k operations and keep track of the element that reduces result to minimum.

C++

```
 // CPP program to find minimum sum of product
// of two arrays with k operations allowed on
// first array.
#include <bits/stdc++.h>
using namespace std;

// Function to find the minimum product
int minproduct(int a[], int b[], int n, int k)
{
    int diff = 0, res = 0;
    int temp;
    for (int i = 0; i < n; i++) {

        // Find product of current elements and update
        // result.
        int pro = a[i] * b[i];
        res = res + pro;

        // If both product and b[i] are negative,
        // we must increase value of a[i] to minimize
        // result.
        if (pro < 0 && b[i] < 0)
            temp = (a[i] + 2 * k) * b[i];

        // If both product and a[i] are negative,
        // we must decrease value of a[i] to minimize
        // result.
        else if (pro < 0 && a[i] < 0)
            temp = (a[i] - 2 * k) * b[i];

        // Similar to above two cases for positive
```

```
        // product.
        else if (pro > 0 && a[i] < 0)
            temp = (a[i] + 2 * k) * b[i];
        else if (pro > 0 && a[i] > 0)
            temp = (a[i] - 2 * k) * b[i];

        // Check if current difference becomes higher
        // than the maximum difference so far.
        int d = abs(pro - temp);
        if (d > diff)
            diff = d;
    }

    return res - diff;
}

// Driver function
int main()
{
    int a[] = { 2, 3, 4, 5, 4 };
    int b[] = { 3, 4, 2, 3, 2 };
    int n = 5, k = 3;
    cout << minproduct(a, b, n, k)
         << endl;
    return 0;
}
```

Java

```
 // Java program to find minimum sum
// of product of two arrays with k
// operations allowed on first array.
import java.math.*;

class GFG {

// Function to find the minimum product
static int minproduct(int a[], int b[], int n,
                                        int k)
{
    int diff = 0, res = 0;
    int temp = 0;
    for (int i = 0; i < n; i++) {

        // Find product of current elements
        // and update result.
        int pro = a[i] * b[i];
        res = res + pro;
```

```
        // If both product and b[i] are
        // negative, we must increase value
        // of a[i] to minimize result.
        if (pro < 0 && b[i] < 0)
            temp = (a[i] + 2 * k) * b[i];

        // If both product and a[i] are
        // negative, we must decrease value
        // of a[i] to minimize result.
        else if (pro < 0 && a[i] < 0)
            temp = (a[i] - 2 * k) * b[i];

        // Similar to above two cases
        // for positive product.
        else if (pro > 0 && a[i] < 0)
            temp = (a[i] + 2 * k) * b[i];
        else if (pro > 0 && a[i] > 0)
            temp = (a[i] - 2 * k) * b[i];

        // Check if current difference
        // becomes higher than the maximum
        // difference so far.
        int d = Math.abs(pro - temp);
        if (d > diff)
            diff = d;
    }

    return res - diff;
}

// Driver function
public static void main(String[] args)
{
    int a[] = { 2, 3, 4, 5, 4 };
    int b[] = { 3, 4, 2, 3, 2 };
    int n = 5, k = 3;
    System.out.println(minproduct(a, b, n, k));
}
}

// This code is contributed by Prerna Saini
```

Python3

```
 # Python program to find
# minimum sum of product
# of two arrays with k
```

```
# operations allowed on
# first array.

# Function to find the minimum product
def minproduct(a,b,n,k):

    diff = 0
    res = 0
    for i in range(n):

        # Find product of current
        # elements and update result.
        pro = a[i] * b[i]
        res = res + pro

        # If both product and
        # b[i] are negative,
        # we must increase value
        # of a[i] to minimize result.
        if (pro < 0 and b[i] < 0):
            temp = (a[i] + 2 * k) * b[i]

        # If both product and
        # a[i] are negative,
        # we must decrease value
        # of a[i] to minimize result.
        elif (pro < 0 and a[i] < 0):
            temp = (a[i] - 2 * k) * b[i]

        # Similar to above two cases
        # for positive product.
        elif (pro > 0 and a[i] < 0):
            temp = (a[i] + 2 * k) * b[i]
        elif (pro > 0 and a[i] > 0):
            temp = (a[i] - 2 * k) * b[i]

        # Check if current difference
        # becomes higher
        # than the maximum difference so far.
        d = abs(pro - temp)

        if (d > diff):
            diff = d
    return res - diff

# Driver function
a = [ 2, 3, 4, 5, 4 ]
b = [ 3, 4, 2, 3, 2 ]
```

```
n = 5
k = 3

print(minproduct(a, b, n, k))

# This code is contributed
# by Azkia Anam.
```

C#

```
 // C# program to find minimum sum
// of product of two arrays with k
// operations allowed on first array.
using System;

class GFG {

    // Function to find the minimum product
    static int minproduct(int []a, int []b,
                                int n, int k)
    {
        int diff = 0, res = 0;
        int temp = 0;
        for (int i = 0; i < n; i++)
        {

            // Find product of current elements
            // and update result.
            int pro = a[i] * b[i];
            res = res + pro;

            // If both product and b[i] are
            // negative, we must increase value
            // of a[i] to minimize result.
            if (pro < 0 && b[i] < 0)
                temp = (a[i] + 2 * k) * b[i];

            // If both product and a[i] are
            // negative, we must decrease value
            // of a[i] to minimize result.
            else if (pro < 0 && a[i] < 0)
                temp = (a[i] - 2 * k) * b[i];

            // Similar to above two cases
            // for positive product.
            else if (pro > 0 && a[i] < 0)
                temp = (a[i] + 2 * k) * b[i];
            else if (pro > 0 && a[i] > 0)
```

```
                temp = (a[i] - 2 * k) * b[i];

            // Check if current difference
            // becomes higher than the maximum
            // difference so far.
            int d = Math.Abs(pro - temp);
            if (d > diff)
                diff = d;
        }

        return res - diff;
    }

    // Driver function
    public static void Main()
    {
        int []a = { 2, 3, 4, 5, 4 };
        int []b = { 3, 4, 2, 3, 2 };
        int n = 5, k = 3;

        Console.WriteLine(minproduct(a, b, n, k));
    }
}

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

PHP

```
 <?php
// PHP program to find minimum sum of product
// of two arrays with k operations allowed on
// first array.

// Function to find the minimum product
function minproduct( $a, $b, $n, $k)
{
    $diff = 0; $res = 0;
    $temp;
    for ( $i = 0; $i < $n; $i++) {

        // Find product of current
        // elements and update
        // result.
        $pro = $a[$i] * $b[$i];
        $res = $res + $pro;

        // If both product and b[i]
        // are negative, we must
```

```
        // increase value of a[i]
        // to minimize result.
        if ($pro < 0 and $b[$i] < 0)
            $temp = ($a[$i] + 2 * $k) *
                                $b[$i];

        // If both product and
        // a[i] are negative,
        // we must decrease value
        // of a[i] to minimize
        // result.
        else if ($pro < 0 and $a[$i] < 0)
            $temp = ($a[$i] - 2 * $k) * $b[$i];

        // Similar to above two
        // cases for positive
        // product.
        else if ($pro > 0 and $a[$i] < 0)
            $temp = ($a[$i] + 2 * $k) * $b[$i];
        else if ($pro > 0 and $a[$i] > 0)
            $temp = ($a[$i] - 2 * $k) * $b[$i];

        // Check if current difference becomes higher
        // than the maximum difference so far.
        $d = abs($pro - $temp);
        if ($d > $diff)
            $diff = $d;
    }

    return $res - $diff;
}

    // Driver Code
    $a = array(2, 3, 4, 5, 4 ,0);
    $b =array(3, 4, 2, 3, 2);
    $n = 5;
    $k = 3;
    echo minproduct($a, $b, $n, $k);

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

Output :

```
25
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/minimum-sum-product-two-arrays/

Chapter 101

Minimum sum of two numbers formed from digits of an array

Minimum sum of two numbers formed from digits of an array - GeeksforGeeks

Given an array of digits (values are from 0 to 9), find the minimum possible sum of two numbers formed from digits of the array. All digits of given array must be used to form the two numbers.

Examples:

```
Input: [6, 8, 4, 5, 2, 3]
Output: 604
The minimum sum is formed by numbers
358 and 246

Input: [5, 3, 0, 7, 4]
Output: 82
The minimum sum is formed by numbers
35 and 047
```

Since we want to minimize the sum of two numbers to be formed, we must divide all digits in two halves and assign half-half digits to them. We also need to make sure that the leading digits are smaller.
We build a Min Heap with the elements of the given array, which takes O(n) worst time. Now we retrieve min values (2 at a time) of array, by polling from the Priority Queue and append these two min values to our numbers, till the heap becomes empty, i.e., all the elements of array get exhausted. We return the sum of two formed numbers, which is our required answer.
C/C++

```
// C++ program to find minimum sum of two numbers
```

```
// formed from all digits in a given array.
#include<bits/stdc++.h>
using namespace std;

// Returns sum of two numbers formed
// from all digits in a[]
int minSum(int arr[], int n)
{
    // min Heap
    priority_queue <int, vector<int>, greater<int> > pq;

    // to store the 2 numbers formed by array elements to
    // minimize the required sum
    string num1, num2;

    // Adding elements in Priority Queue
    for(int i=0; i<n; i++)
        pq.push(arr[i]);

    // checking if the priority queue is non empty
    while(!pq.empty())
    {
        // appending top of the queue to the string
        num1+=(48 + pq.top());
        pq.pop();
        if(!pq.empty())
        {
            num2+=(48 + pq.top());
            pq.pop();
        }
    }

    // converting string to integer
    int a = atoi(num1.c_str());
    int b = atoi(num2.c_str());

    // returning the sum
    return a+b;
}

int main()
{
    int arr[] = {6, 8, 4, 5, 2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout<<minSum(arr, n)<<endl;
    return 0;
}
// Contributed By: Harshit Sidhwa
```

Java

```
 // Java program to find minimum sum of two numbers
// formed from all digits in a given array.
import java.util.PriorityQueue;

class MinSum
{
    // Returns sum of two numbers formed
    // from all digits in a[]
    public static long solve(int[] a)
    {
        // min Heap
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();

        // to store the 2 numbers formed by array elements to
        // minimize the required sum
        StringBuilder num1 = new StringBuilder();
        StringBuilder num2 = new StringBuilder();

        // Adding elements in Priority Queue
        for (int x : a)
            pq.add(x);

        // checking if the priority queue is non empty
        while (!pq.isEmpty())
        {
            num1.append(pq.poll()+ "");
            if (!pq.isEmpty())
                num2.append(pq.poll()+ "");
        }

        // the required sum calculated
        long sum = Long.parseLong(num1.toString()) +
                   Long.parseLong(num2.toString());

        return sum;
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = {6, 8, 4, 5, 2, 3};
        System.out.println("The required sum is "+ solve(arr));
    }
}
```

Output:

```
The required sum is 604
```

Source

https://www.geeksforgeeks.org/minimum-sum-two-numbers-formed-digits-array-2/

Chapter 102

Number of chocolates left after k iterations

Number of chocolates left after k iterations - GeeksforGeeks

Given a pile of chocolates and an integer 'k' i.e. the number of iterations, the task is to find the number of chocolates left after k iterations.
Note: In every iteration, we can choose a pile with a maximum number of chocolates, after that square root of chocolate remains and rest is eaten.
Examples:

```
Input: Chocolates = 100000000, Iterations = 3
Output: 10

Input: Chocolates = 200, Iterations = 2
Output: 4
```

Note: Output is printed after rounding off the value as in the 2nd example, the output will be around **3.76 approx.**
Approach:
It is given that maximum no. of chocolates are selected so consider total pile since it will be maximum.
Next, it is given that in the each iteration only square of chocolates are left so that by considering the mathematics equation of

```
(((number)n)n)...n for k times = (number)nk
```

Since here k times the square root is performed so the $(1/2)^k$ is powered with the N.

Consider the example of 100000000 chocolates and no. of iterations is 3 then it will be as

```
(((100000000)1/2)1/2)1/2 = (100000000)(1/2)3 = 10
```

Below is the required formula to find the remaining chocolates:

```
round(pow(n, (1.0/pow(2, k))))
```

C++

```
 // C++ program to find remaining
// chocolates after k iterations
#include <bits/stdc++.h>
using namespace std;

// Function to find the chocolates left
int results(int n, int k)
{
    return round(pow(n, (1.0 / pow(2, k))));
}

// Driver code
int main()
{
    int k = 3, n = 100000000;

    cout << "Chocolates left after " << k
         << " iterations are " << results(n, k);

    return 0;
}
```

Java

```
 // Java program to find remaining
// chocolates after k iterations
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
// Function to find the
// chocolates left
static int results(int n, int k)
{
```

```
    return (int)Math.round(Math.pow(n,
           (1.0 / Math.pow(2.0, k))));
}

// Driver code
public static void main(String args[])
{
    int k = 3, n = 100000000;

    System.out.print("Chocolates left after " + k +
               " iterations are " + results(n, k));
}
}

// This code is contributed by Subhadeep
```

C#

```
 // C# program to find remaining
// chocolates after k iterations
using System;

class GFG
{
// Function to find the
// chocolates left
static int results(int n, int k)
{
    return (int)Math.Round(Math.Pow(n,
           (1.0 / Math.Pow(2.0, k))));
}

// Driver code
public static void Main()
{
    int k = 3, n = 100000000;

    Console.Write("Chocolates left after " +
                   k + " iterations are " +
                            results(n, k));
}
}

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

Python

```
 # Python program to find
# remaining chocolates
# after k iterations

# Function to find the
# chocolates left
def results(n, k):

    return round(pow(n, (1.0 /
                 pow(2, k))))

# Driver code
k = 3
n = 100000000

print("Chocolates left after"),
print(k),
print("iterations are"),
print(int(results(n, k)))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// PHP program to find remaining
// chocolates after k iterations

// Function to find
// the chocolates left
function results($n, $k)
{
    return round(pow($n, (1.0 /
                 pow(2, $k))));
}

// Driver code
$k = 3;
$n = 100000000;
echo ("Chocolates left after ");
echo ($k);

echo (" iterations are ");
echo (results($n, $k));

// This code is contributed
// by Shivi_Aggarwal
```

```
?>
```

Output:

```
Chocolates left after 3 iterations are 10
```

Improved By : Shivi_Aggarwal, tufan_gupta2000, ChitraNayal

Source

https://www.geeksforgeeks.org/number-of-chocolates-left-after-k-iterations/

Chapter 103

Number of single cycle components in an undirected graph

Number of single cycle components in an undirected graph - GeeksforGeeks

Given a set of 'n' vertices and 'm' edges of an undirected simple graph (no parallel edges and no self-loop), find the number of single-cycle-components present in the graph. A single-cyclic-component is a graph of n nodes containing a single cycle through all nodes of the component.

Example:

```
Let us consider the following graph with 15 vertices.


Input: V = 15, E = 14
       1 10  // edge 1
       1 5   // edge 2
       5 10  // edge 3
       2 9   // ..
       9 15  // ..
       2 15  // ..
       2 12  // ..
       12 15 // ..
       13 8  // ..
       6 14  // ..
       14 3  // ..
       3 7   // ..
       7 11  // edge 13
       11 6  // edge 14
```

```
Output :2
In the above-mentioned example, the two
single-cyclic-components are composed of
vertices (1, 10, 5) and (6, 11, 7, 3, 14)
respectively.
```

Now we can easily see that a single-cycle-component is a connected component where every vertex has the degree as two.
Therefore, in order to solve this problem we first identify all the connected components of the disconnected graph. For this, we use depth-first search algorithm. For the DFS algorithm to work, it is required to maintain an array 'found' to keep an account of all the vertices that have been discovered by the recursive function DFS. Once all the elements of a particular connected component are discovered (like vertices(9, 2, 15, 12) form a connected graph component), we check if all the vertices in the component are having the degree equal to two. If yes, we increase the counter variable 'count' which denotes the number of single-cycle-components found in the given graph. To keep an account of the component we are presently dealing with, we may use a vector array 'curr_graph' as well.

```
 // CPP program to find single cycle components
// in a graph.
#include <bits/stdc++.h>
using namespace std;

const int N = 100000;

// degree of all the vertices
int degree[N];

// to keep track of all the vertices covered
// till now
bool found[N];

// all the vertices in a particular
// connected component of the graph
vector<int> curr_graph;

// adjacency list
vector<int> adj_list[N];

// depth-first traversal to identify all the
// nodes in a particular connected graph
// component
void DFS(int v)
{
    found[v] = true;
    curr_graph.push_back(v);
    for (int it : adj_list[v])
        if (!found[it])
```

```
            DFS(it);
}

// function to add an edge in the graph
void addEdge(vector<int> adj_list[N], int src,
             int dest)
{
    // for index decrement both src and dest.
    src--, dest--;
    adj_list[src].push_back(dest);
    adj_list[dest].push_back(src);
    degree[src]++;
    degree[dest]++;
}

int countSingleCycles(int n, int m)
{
    // count of cycle graph components
    int count = 0;
    for (int i = 0; i < n; ++i) {
        if (!found[i]) {
            curr_graph.clear();
            DFS(i);

            // traversing the nodes of the
            // current graph component
            int flag = 1;
            for (int v : curr_graph) {
                if (degree[v] == 2)
                    continue;
                else {
                    flag = 0;
                    break;
                }
            }
            if (flag == 1) {
                count++;
            }
        }
    }
    return(count);
}

int main()
{
    // n->number of vertices
    // m->number of edges
    int n = 15, m = 14;
```

```
    addEdge(adj_list, 1, 10);
    addEdge(adj_list, 1, 5);
    addEdge(adj_list, 5, 10);
    addEdge(adj_list, 2, 9);
    addEdge(adj_list, 9, 15);
    addEdge(adj_list, 2, 15);
    addEdge(adj_list, 2, 12);
    addEdge(adj_list, 12, 15);
    addEdge(adj_list, 13, 8);
    addEdge(adj_list, 6, 14);
    addEdge(adj_list, 14, 3);
    addEdge(adj_list, 3, 7);
    addEdge(adj_list, 7, 11);
    addEdge(adj_list, 11, 6);

    cout << countSingleCycles(n, m);

    return 0;
}
```

Output:

```
2
```

Hence, total number of cycle graph component is found.

Improved By : PiyushKumar

Source

https://www.geeksforgeeks.org/number-of-simple-cyclic-components-in-an-undirected-graph/

Chapter 104

Operating System Program for Next Fit algorithm in Memory Management

Operating System Program for Next Fit algorithm in Memory Management - GeeksforGeeks

Prerequisite: Partition allocation methods

What is Next Fit ?

Next fit is a modified version of 'first fit'. It begins as first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer roves along the memory chain to search for a next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free block chain.

What are its advantage over first fit ?

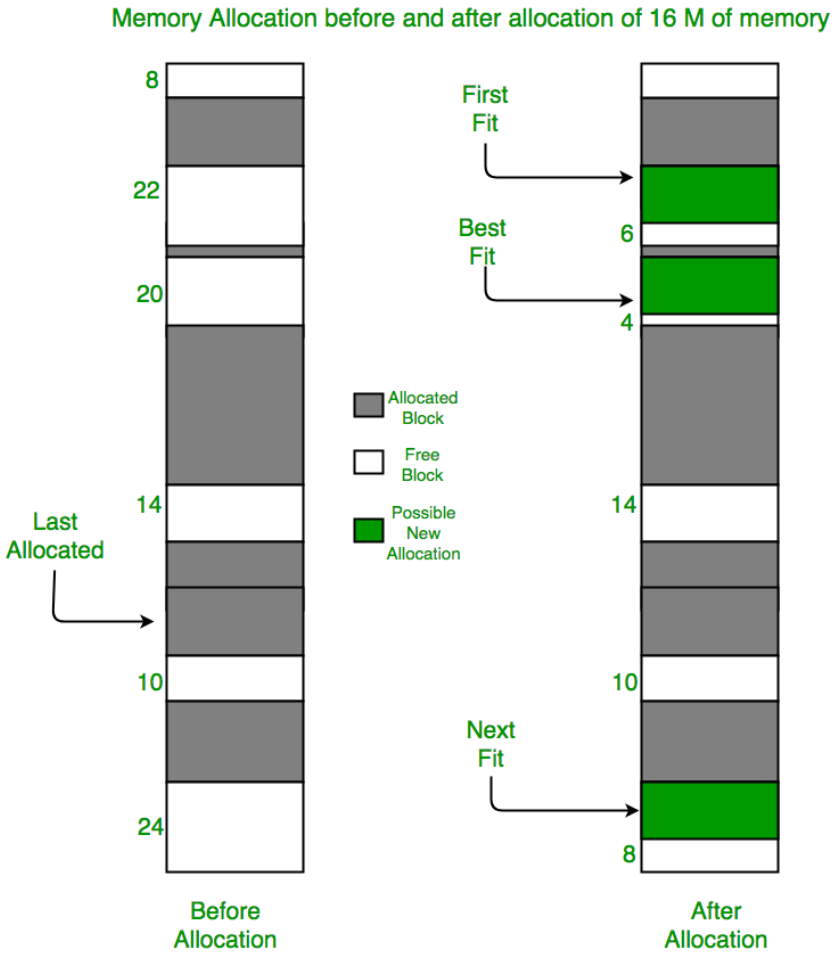
- First fit is a straight and fast algorithm, but tends to cut large portion of free parts into small pieces due to which, processes that needs large portion of memory block would not get anything even if the sum of all small pieces is greater than it required which is so called external fragmentation problem.
- Another problem of first fit is that it tends to allocate memory parts at the begining of the memory, which may leads to more internal fragements at the begining. Next fit tries to address this problem by starting search for the free portion of parts not from the start of the memory, but from where it ends last time.
- Next fit is a very fast searching algorithm and is also comparatively faster than First Fit and Best Fit Memory Management Algorithms.

```
Example:
Input :  blockSize[] = {5, 10, 20};
```

```
    processSize[] = {10, 20, 30};
Output:
Process No.     Process Size    Block no.
 1              10              2
 2              20              3
 3              30              Not Allocated
```

Algorithm:

1. Input the number of memory blocks and their sizes and initializes all the blocks as free.
2. Input the number of processes and their sizes.
3. Start by picking each process and check if it can be assigned to current block, if yes, allocate it the required memory and check for next process but from the block where we left not from starting.
4. If current block size is smaller then keep checking the further blocks.




```
 // C/C++ program for next fit
// memory management algorithm
#include <bits/stdc++.h>
using namespace std;

// Function to allocate memory to blocks as per Next fit
// algorithm
void NextFit(int blockSize[], int m, int processSize[], int n)
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n], j = 0;

    // Initially no block is assigned to any process
```

```
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i = 0; i < n; i++) {

        // Do not start from beginning
        while (j < m) {

            if (blockSize[j] >= processSize[i]) {

                // allocate block j to p[i] process
                allocation[i] = j;

                // Reduce available memory in this block.
                blockSize[j] -= processSize[i];

                break;
            }

            // mod m will help in traversing the blocks from
            // starting block after we reach the end.
            j = (j + 1) % m;
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++) {
        cout << " " << i + 1 << "\t\t" << processSize[i]
             << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

// Driver program
int main()
{
    int blockSize[] = { 5, 10, 20 };
    int processSize[] = { 10, 20, 5 };
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    NextFit(blockSize, m, processSize, n);
```

```
    return 0;
}
```

Output:

```
Process No.    Process Size    Block no.
 1               10              2
 2               20              3
 3               5               1
```

Improved By : ishita_thakkar

Source

https://www.geeksforgeeks.org/program-next-fit-algorithm-memory-management/

Chapter 105

Optimal Storage on Tapes

Optimal Storage on Tapes - GeeksforGeeks

Given n programs stored on a computer tape and length of each program i is L_i where [illegible], find the order in which the programs should be stored in the tape for which the Mean Retrieval Time (MRT given as [illegible]) is minimized.

Example:

```
Input : n = 3
        L[] = { 5, 3, 10 }
Output : Order should be { 3, 5, 10 } with MRT = 29/3
```

Prerequisites: Magnetic Tapes Data Storage

Let us first break down the problem and understand what needs to be done.

A magnetic tape provides only sequential access of data. In an audio tape/cassette, unlike a CD, a fifth song from the tape can't be just directly played. The length of the first four songs must be traversed to play the fifth song. So in order to access certain data, head of the tape should be positioned accordingly.

Now suppose there are 4 songs in a tape of audio lengths 5, 7, 3 and 2 mins respectively. In order to play the fourth song, we need to traverse an audio length of 5 + 7 + 3 = 15 mins and then position the tape head.
Retrieval time of the data is the time taken to retrieve/access that data in its entirety. Hence retrieval time of the fourth song is 15 + 2 = 17 mins.

Now, considering that all programs in a magnetic tape are retrieved equally often and the tape head points to the front of the tape every time, a new term can be defined called the Mean Retrieval Time (MRT).

Let's suppose that the retrieval time of program i is T_i. Therefore,

MRT is the average of all such T_i. Therefore , or

The sequential access of data in a tape has some limitations. Order must be defined in which the data/programs in a tape are stored so that least MRT can be obtained. Hence the order of storing becomes very important to reduce the data retrieval/access time.
Thus, the task gets reduced – to define the correct order and hence minimize the MRT, i.e. to minimize the term

For e.g. Suppose there are 3 programs of lengths 2, 5 and 4 respectively. So there are total $3! = 6$ possible orders of storage.

	Order	Total Retrieval Time	Mean Retrieval Time
1	1 2 3	$2 + (2 + 5) + (2 + 5 + 4) = 20$	20/3
2	1 3 2	$2 + (2 + 4) + (2 + 4 + 5) = 19$	19/3
3	2 1 3	$5 + (5 + 2) + (5 + 2 + 4) = 23$	23/3
4	2 3 1	$5 + (5 + 4) + (5 + 4 + 2) = 25$	25/3
5	3 1 2	$4 + (4 + 2) + (4 + 2 + 5) = 21$	21/3
6	3 2 1	$4 + (4 + 5) + (4 + 5 + 2) = 24$	24/3

It's clear that by following the second order in storing the programs, the mean retrieval time is least.

In above example, the first program's length is added 'n' times, the second 'n-1' times...and so on till the last program is added only once. So, careful analysis suggests that in order to minimize the MRT, programs having greater lengths should be put towards the end so that the summation is reduced. Or, the lengths of the programs should be sorted in increasing order. That's the **Greedy Algorithm** in use – at each step we make the immediate choice of putting the program having the least time first, in order to build up the ultimate optimized solution to the problem piece by piece.

Below is the implementation:
C++

```
 // CPP Program to find the order
// of programs for which MRT is
// minimized
#include <bits/stdc++.h>

using namespace std;
```

```
// This functions outputs the required
// order and Minimum Retrieval Time
void findOrderMRT(int L[], int n)
{
    // Here length of i'th program is L[i]
    sort(L, L + n);

    // Lengths of programs sorted according to increasing
    // lengths. This is the order in which the programs
    // have to be stored on tape for minimum MRT
    cout << "Optimal order in which programs are to be"
            "stored is: ";
    for (int i = 0; i < n; i++)
        cout << L[i] << " ";
    cout << endl;

    // MRT - Minimum Retrieval Time
    double MRT = 0;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = 0; j <= i; j++)
            sum += L[j];
        MRT += sum;
    }
    MRT /= n;
    cout << "Minimum Retrieval Time of this"
           " order is " << MRT;
}

// Driver Code to test above function
int main()
{
    int L[] = { 2, 5, 4 };
    int n = sizeof(L) / sizeof(L[0]);
    findOrderMRT(L, n);
    return 0;
}
```

Java

```
 // Java Program to find the order
// of programs for which MRT is
// minimized
import java.io.*;
import java .util.*;

class GFG
{
```

```
// This functions outputs
// the required order and
// Minimum Retrieval Time
static void findOrderMRT(int []L,
                         int n)
{
    // Here length of
    // i'th program is L[i]
    Arrays.sort(L);

    // Lengths of programs sorted
    // according to increasing lengths.
    // This is the order in which
    // the programs have to be stored
    // on tape for minimum MRT
    System.out.print("Optimal order in which " +
              "programs are to be stored is: ");
    for (int i = 0; i < n; i++)
        System.out.print(L[i] + " ");
        System.out.println();

    // MRT - Minimum Retrieval Time
    double MRT = 0;
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = 0; j <= i; j++)
            sum += L[j];
        MRT += sum;
    }
    MRT /= n;
    System.out.print( "Minimum Retrieval Time" +
                    " of this order is " + MRT);
}

// Driver Code
public static void main (String[] args)
{
    int []L = { 2, 5, 4 };
    int n = L.length;
    findOrderMRT(L, n);
}
}

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

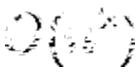
C#

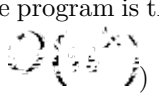
```
 // C# Program to find the
// order of programs for
// which MRT is minimized
using System;

class GFG
{

// This functions outputs
// the required order and
// Minimum Retrieval Time
static void findOrderMRT(int []L,
                         int n)
{
    // Here length of
    // i'th program is L[i]
    Array.Sort(L);

    // Lengths of programs sorted
    // according to increasing lengths.
    // This is the order in which
    // the programs have to be stored
    // on tape for minimum MRT
    Console.Write("Optimal order in " +
                  "which programs are" +
                  " to be stored is: ");
    for (int i = 0; i < n; i++)
        Console.Write(L[i] + " ");
        Console.WriteLine();

    // MRT - Minimum Retrieval Time
    double MRT = 0;
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = 0; j <= i; j++)
            sum += L[j];
        MRT += sum;
    }
    MRT /= n;
    Console.WriteLine("Minimum Retrieval " +
                  "Time of this order is " +
                                       MRT);
}

// Driver Code
```

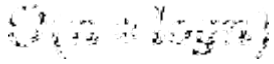
```
public static void Main ()
{
    int []L = { 2, 5, 4 };
    int n = L.Length;
    findOrderMRT(L, n);
}
}

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

Output:

```
Optimal order in which programs are to be stored are: 2 4 5
Minimum Retrieval Time of this order is 6.33333
```

Time complexity of the above program is the time complexity for sorting, that is 

(Since std::sort() operates in )

But the time complexity can be reduced to  by avoiding two loops and can be done in this manner.

```
for (int i = 0; i < n; i++)
    MRT += (n - i) * L[i];
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/optimal-storage-tapes/

Chapter 106

Pair formation such that maximum pair sum is minimized

Pair formation such that maximum pair sum is minimized - GeeksforGeeks

Given an array of size 2 * N integers. Divide the array into N pairs, such that the maximum pair sum is minimized. In other words, the optimal division of array into N pairs should result into a maximum pair sum which is minimum of other maximum pair sum of all possibilities.

Examples:

> Input : N = 2
> arr[] = { 5, 8, 3, 9 }
> Output : (3, 9) (5, 8)
> **Explanation:**
> Possible pairs are :
> 1. (8, 9) (3, 5) Maximum Sum of a Pair = 17
> 2. (5, 9) (3, 8) Maximum Sum of a Pair = 14
> 3. (3, 9) (5, 8) Maximum Sum of a Pair = 13
> Thus, in case 3, the maximum pair sum is minimum of all the other cases. Hence, the answer is(3, 9) (5, 8).
>
> Input : N = 2
> arr[] = { 9, 6, 5, 1 }
> Output : (1, 9) (5, 6)

Approach: The idea is to first sort the given array and then iterate over the loop to form pairs (i, j) where i would start from 0 and j would start from end of array correspondingly. Increment i and Decrement j to form the next pair and so on.

Below is the implementation of above approach.
C++

```
 // CPP Program to divide the array into
// N pairs such that maximum pair is minimized
#include <bits/stdc++.h>

using namespace std;

void findOptimalPairs(int arr[], int N)
{
    sort(arr, arr + N);

    // After Sorting Maintain two variables i and j
    // pointing to start and end of array Such that
    // smallest element of array pairs with largest
    // element
    for (int i = 0, j = N - 1; i <= j; i++, j--)
        cout << "(" << arr[i] << ", " << arr[j] << ")" << " ";
}

// Driver Code
int main()
{
    int arr[] = { 9, 6, 5, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);

    findOptimalPairs(arr, N);
    return 0;
}
```

Java

```
 // Java Program to divide the array into
// N pairs such that maximum pair is minimized
import java.io.*;
import java.util.Arrays;

class GFG {

static void findOptimalPairs(int arr[], int N)
{
    Arrays.sort(arr);

    // After Sorting Maintain two variables i and j
    // pointing to start and end of array Such that
    // smallest element of array pairs with largest
    // element
    for (int i = 0, j = N - 1; i <= j; i++, j--)
        System.out.print( "(" + arr[i] + ", " + arr[j] + ")" + " ");
}
```

```
    // Driver Code
    public static void main (String[] args)
    {
        int arr[] = {9, 6, 5, 1};
        int N = arr.length;

        findOptimalPairs(arr, N);
    }
}

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

Output:

```
(1, 9) (5, 6)
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/pair-formation-maximum-pair-sum-minimized/

Chapter 107

Paper Cut into Minimum Number of Squares

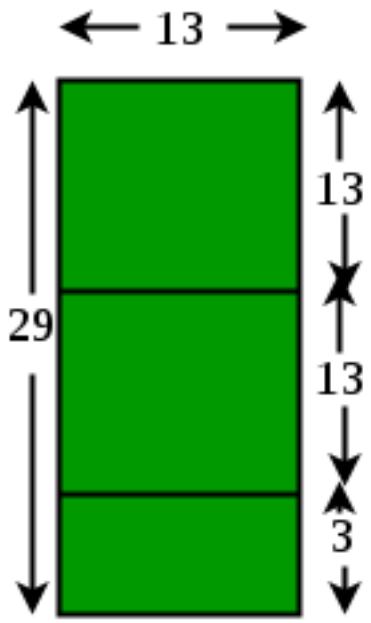
Paper Cut into Minimum Number of Squares - GeeksforGeeks

Given a paper of size A x B. Task is to cut the paper into squares of any size. Find the minimum number of squares that can be cut from the paper.

Examples:

```
Input  : 13 x 29
Output : 9
Explanation :
2 (squares of size 13x13) +
4 (squares of size 3x3) +
3 (squares of size 1x1)=9

Input  : 4 x 5
Output : 5
Explanation :
1 (squares of size 4x4) +
4 (squares of size 1x1)
```

We know that if we want to cut minimum number of squares from the paper then we would have to cut largest square possible from the paper first and largest square will have same side as smaller side of the paper. For example if paper have the size 13 x 29, then maximum square will be of side 13. so we can cut 2 square of size 13 x 13 (29/13 = 2). Now remaining paper will have size 3 x 13. Similarly we can cut remaining paper by using 4 squares of size 3 x 3 and 3 squares of 1 x 1. So minimum 9 squares can be cut from the Paper of size 13 x 29.



Below is the implementation of above approach.
C++

```
 // C++ program to find minimum number of squares
// to cut a paper.
#include<bits/stdc++.h>
using namespace std;

// Returns min number of squares needed
int minimumSquare(int a, int b)
{
    long long result = 0, rem = 0;

    // swap if a is small size side .
    if (a < b)
        swap(a, b);

    // Iterate until small size side is
    // greater then 0
    while (b > 0)
    {
        // Update result
        result += a/b;

        long long rem = a % b;
        a = b;
        b = rem;
```

```
    }

    return result;
}

// Driver code
int main()
{
    int n = 13, m = 29;
    cout << minimumSquare(n, m);
    return 0;
}
```

Java

```
 // Java program to find minimum
// number of squares to cut a paper.
class GFG{

// To swap two numbers
static void swap(int a,int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Returns min number of squares needed
static int minimumSquare(int a, int b)
{
    int result = 0, rem = 0;

    // swap if a is small size side .
    if (a < b)
        swap(a, b);

    // Iterate until small size side is
    // greater then 0
    while (b > 0)
    {
        // Update result
        result += a/b;
        rem = a % b;
        a = b;
        b = rem;
    }

    return result;
```

```
}

// Driver code
public static void main(String[] args)
{
    int n = 13, m = 29;
    System.out.println(minimumSquare(n, m));
}
}

//This code is contributed by Smitha Dinesh Semwal.
```

Python3

```
 # Python 3 program to find minimum
# number of squares to cut a paper.

# Returns min number of squares needed
def minimumSquare(a, b):

    result = 0
    rem = 0

    # swap if a is small size side .
    if (a < b):
        a, b = b, a

    # Iterate until small size side is
    # greater then 0
    while (b > 0):

        # Update result
        result += int(a / b)

        rem = int(a % b)
        a = b
        b = rem

    return result

# Driver code
n = 13
m = 29

print(minimumSquare(n, m))

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

Output:

```
9
```

Note that the above Greedy solution doesn't always produce optimal result. For example if input is 36 x 30, the above algorithm would produce output 6, but we can cut the paper in 5 squares
1) Three squares of size 12 x 12
2) Two squares of size 18 x 18.

Thanks to Sergey V. Pereslavtsev for pointing out the above case.

Source

https://www.geeksforgeeks.org/paper-cut-minimum-number-squares/

Chapter 108

Partition into two subarrays of lengths k and (N – k) such that the difference of sums is maximum

Partition into two subarrays of lengths k and (N - k) such that the difference of sums is maximum - GeeksforGeeks

Given an array of non-negative integers of length N and an integer k. Partition the given array into two subarrays of length K and N – k so that the difference between the sum of both subarray is maximum.

Examples :

```
Input : arr[] = {8, 4, 5, 2, 10}
        k = 2
Output : 17
Explanation :
Here, we can make first subarray of length k = {4, 2}
and second subarray of length N - k = {8, 5, 10}. Then,
the max_difference = (8 + 5 + 10) - (4 + 2) = 17.

Input : arr[] = {1, 1, 1, 1, 1, 1, 1, 1}
        k = 3
Output : 2
Explanation :
Here, subarrays would be {1, 1, 1, 1, 1} and {1, 1, 1}.
So, max_difference would be 2.
```

Choose k numbers with largest possible sum. Then the solution obviously is k largest numbers. So that here greedy algorithm works – at each step we choose the largest possible number until we get all K numbers.

In this problem we should divide the array of N numbers into two subarrays of k and $N - k$ numbers respectively. Consider two cases –

- The subarray with largest sum, among these two subarrays, is subarray of K numbers. Then we want to maximize the sum in it, since the sum in the second subarray will only decrease if the sum in the first subarray will increase. So we are now in sub-problem considered above and should choose k largest numbers.
- The subarray with largest sum, among these two subarray, is subarray of N – k numbers. Similarly to the previous case we then have to choose N – k largest numbers among all numbers.

Now, Let's think which of the two above cases actually gives the answer. We can easily see that larger difference would be when more numbers are included to the group of largest numbers. Hence we could set M = max(k, N – k), find the sum of M largest numbers (let it be S1) and then the answer is S1 – (S – S1), where S is the sum of all numbers.

Below is the implementation of the above approach :
C++

```
 // CPP program to calculate max_difference between
// the sum of two subarrays of length k and N - k
#include <bits/stdc++.h>
using namespace std;

// Function to calculate max_difference
int maxDifference(int arr[], int N, int k)
{
    int M, S = 0, S1 = 0, max_difference = 0;

    // Sum of the array
    for (int i = 0; i < N; i++)
        S += arr[i];

    // Sort the array in descending order
    sort(arr, arr + N, greater<int>());
    M = max(k, N - k);
    for (int i = 0; i < M; i++)
        S1 += arr[i];

    // Calculating max_difference
    max_difference = S1 - (S - S1);
    return max_difference;
}
```

```
// Driver function
int main()
{
    int arr[] = { 8, 4, 5, 2, 10 };
    int N = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    cout << maxDifference(arr, N, k) << endl;
    return 0;
}
```

Python3

```
 # Python3 code to calculate max_difference
# between the sum of two subarrays of
# length k and N - k

# Function to calculate max_difference
def maxDifference(arr, N, k ):
    S = 0
    S1 = 0
    max_difference = 0

    # Sum of the array
    for i in range(N):
        S += arr[i]

    # Sort the array in descending order
    arr.sort(reverse=True)
    M = max(k, N - k)
    for i in range( M):
        S1 += arr[i]

    # Calculating max_difference
    max_difference = S1 - (S - S1)
    return max_difference

# Driver Code
arr = [ 8, 4, 5, 2, 10 ]
N = len(arr)
k = 2
print(maxDifference(arr, N, k))

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

PHP

```
 <?php
```

```
// PHP program to calculate
// max_difference between
// the sum of two subarrays
// of length k and N - k

// Function to calculate
// max_difference
function maxDifference($arr, $N, $k)
{
    $M; $S = 0; $S1 = 0;
    $max_difference = 0;

    // Sum of the array
    for ($i = 0; $i < $N; $i++)
        $S += $arr[$i];

    // Sort the array in
    // descending order
    rsort($arr);
    $M = max($k, $N - $k);
    for ($i = 0; $i < $M; $i++)
        $S1 += $arr[$i];

    // Calculating
    // max_difference
    $max_difference = $S1 - ($S - $S1);
    return $max_difference;
}

// Driver Code
$arr = array(8, 4, 5, 2, 10);
$N = count($arr);
$k = 2;
echo maxDifference($arr, $N, $k);

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

Output :

```
17
```

Further Optimizations : We can use Heap (or priority queue) to find M largest elements efficiently. Refer k largest(or smallest) elements in an array for details.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/partition-into-two-subarrays-of-lengths-k-and-n-k-such-that-the-difference-of-sums

Chapter 109

Place N^2 numbers in matrix such that every row has an equal sum

Place N^2 numbers in matrix such that every row has an equal sum - GeeksforGeeks

Given a number N, place numbers from the range [1, N^2] in an NxN matrix such that sum in every row is equal.

Examples:

```
Input: N = 3
Output: 1 5 9
        2 6 7
        3 4 8
Sum in 1st row: 15
Sum in 2nd row: 15
Sum in 2nd row: 15

Input: N = 5
Output: 1 7 13 19 25
        2 8 14 20 21
        3 9 15 16 22
        4 10 11 17 23
        5 6 12 18 24
```

A **Greedy Approach** has been used to fill the matrix, where the insertion of elements in the matrix is done row-wise. The required matrix can be obtained using below steps:

- Fill the matrix initially with numbers in the range [1, N^2] using matrix traversal.

- Traverse the matrix and change every position in a new matrix considering it as answer matrix by **answer[i][j] = mat[j][(i+j)%n].**

Below is the implementation of the above approach:

```
 // Java program to distribute n^2 numbers to n people
public class Numbers {
    public static int[][] solve(int[][] arr, int n)
    {
        // 2D array for storing the final result
        int[][] ans = new int[n][n];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // using modulo to go to the firs
                // column after the last column
                ans[i][j] = arr[j][(i + j) % n];
            }
        }
        return ans;
    }
    public static void show(int[][] arr, int n)
    {
        int[][] res = solve(arr, n);
        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {
                System.out.print(res[i][j] + " ");
            }
            System.out.println();
        }
    }
    // making a 2D array containing numbers
    public static int[][] makeArray(int n)
    {
        int[][] arr = new int[n][n];
        int c = 1;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                arr[i][j] = c++;
        }
        return arr;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int n = 5;
```

```
        int[][] arr = makeArray(n);
        show(arr, n);
    }
}
```

Output:

```
1 7 13 19 25
2 8 14 20 21
3 9 15 16 22
4 10 11 17 23
5 6 12 18 24
```

Source

https://www.geeksforgeeks.org/place-n2-numbers-in-matrix-such-that-every-row-has-an-equal-sum/

Chapter 110

Policemen catch thieves

Policemen catch thieves - GeeksforGeeks

Given an array of size n that has the following specifications:

1. Each element in the array contains either a policeman or a thief.
2. Each policeman can catch only one thief.
3. A policeman cannot catch a thief who is more than K units away from the policeman.

We need to find the maximum number of thieves that can be caught.

Examples:

```
Input : arr[] = {'P', 'T', 'T', 'P', 'T'},
            k = 1.
Output : 2.
Here maximum 2 thieves can be caught, first
policeman catches first thief and second police-
man can catch either second or third thief.

Input : arr[] = {'T', 'T', 'P', 'P', 'T', 'P'},
            k = 2.
Output : 3.

Input : arr[] = {'P', 'T', 'P', 'T', 'T', 'P'},
            k = 3.
Output : 3.
```

A **brute force** approach would be to check all feasible sets of combinations of police and thief and return the maximum size set among them. Its time complexity is exponential and it can be optimized if we observe an important property.

An **efficient** solution is to use a greedy algorithm. But which greedy property to use can be tricky. We can try using: "For each policeman from the left catch the nearest possible thief." This works for example three given above but fails for example two as it outputs 2 which is incorrect.
We may also try: "For each policeman from the left catch the farthest possible thief". This works for example two given above but fails for example three as it outputs 2 which is incorrect. A symmetric argument can be applied to show that traversing for these from the right side of the array also fails. We can observe that thinking irrespective of the policeman and focusing on just the allotment works:

1. Get the lowest index of policeman p and thief t. Make an allotment if p-t <= k and increment to the next p and t found.
2. Otherwise increment min(p, t) to the next p or t found.
3. Repeat above two steps until next p and t are found.
4. Return the number of allotments made.

Below is the implementation of the above algorithm. It uses vectors to store the indices of police and thief in the array and processes them.
C++

```
 // C++ program to find maximum number of thieves
// caught
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Returns maximum number of thieves that can
// be caught.
int policeThief(char arr[], int n, int k)
{
    int res = 0;
    vector<int> thi;
    vector<int> pol;

    // store indices in the vector
    for (int i = 0; i < n; i++) {
        if (arr[i] == 'P')
            pol.push_back(i);
        else if (arr[i] == 'T')
            thi.push_back(i);
    }

    // track lowest current indices of
    // thief: thi[l], police: pol[r]
    int l = 0, r = 0;
    while (l < thi.size() && r < pol.size()) {
```

```
        // can be caught
        if (abs(thi[l] - pol[r]) <= k) {
            res++;
            l++;
            r++;
        }

        // increment the minimum index
        else if (thi[l] < pol[r])
            l++;
        else
            r++;
    }

    return res;
}

// Driver program
int main()
{
    int k, n;

    char arr1[] = { 'P', 'T', 'T', 'P', 'T' };
    k = 2;
    n = sizeof(arr1) / sizeof(arr1[0]);
    cout << "Maximum thieves caught: "
         << policeThief(arr1, n, k) << endl;

    char arr2[] = { 'T', 'T', 'P', 'P', 'T', 'P' };
    k = 2;
    n = sizeof(arr2) / sizeof(arr2[0]);
    cout << "Maximum thieves caught: "
         << policeThief(arr2, n, k) << endl;

    char arr3[] = { 'P', 'T', 'P', 'T', 'T', 'P' };
    k = 3;
    n = sizeof(arr3) / sizeof(arr3[0]);
    cout << "Maximum thieves caught: "
         << policeThief(arr3, n, k) << endl;

    return 0;
}
```

Java

```
 // Java program to find maximum number of
// thieves caught
import java.util.*;
```

```
import java.io.*;

class GFG
{
    // Returns maximum number of thieves
    // that can be caught.
    static int policeThief(char arr[], int n, int k)
    {
        int res = 0;
        ArrayList<Integer> thi = new ArrayList<Integer>();
        ArrayList<Integer> pol = new ArrayList<Integer>();

        // store indices in the ArrayList
        for (int i = 0; i < n; i++) {
            if (arr[i] == 'P')
            pol.add(i);
            else if (arr[i] == 'T')
            thi.add(i);
        }

        // track lowest current indices of
        // thief: thi[l], police: pol[r]
        int l = 0, r = 0;
        while (l < thi.size() && r < pol.size()) {

            // can be caught
            if (Math.abs(thi.get(l) - pol.get(r)) <= k) {
            res++;
            l++;
            r++;
            }

            // increment the minimum index
            else if (thi.get(l) < pol.get(r))
                l++;
            else
                r++;
        }
        return res;
    }

    // Driver program
    public static void main(String args[])
    {
        int k, n;
        char arr1[] =new char[] { 'P', 'T', 'T',
                                  'P', 'T' };
        k = 2;
```

```
        n = arr1.length;
        System.out.println("Maximum thieves caught: "
                           +policeThief(arr1, n, k));

        char arr2[] =new char[] { 'T', 'T', 'P', 'P',
                                  'T', 'P' };
        k = 2;
        n = arr2.length;
        System.out.println("Maximum thieves caught: "
                           +policeThief(arr2, n, k));

        char arr3[] = new char[]{ 'P', 'T', 'P', 'T',
                                  'T', 'P' };
        k = 3;
        n = arr3.length;
        System.out.println("Maximum thieves caught: "
                           +policeThief(arr3, n, k));
    }
}

/* This code is contributed by Danish kaleem */
```

Output:

```
Maximum thieves caught: 2
Maximum thieves caught: 3
Maximum thieves caught: 3
```

The time complexity of the above approach in **O(N)** where N is the size of the array.

Source

https://www.geeksforgeeks.org/policemen-catch-thieves/

Chapter 111

Practice Questions on Huffman Encoding

Practice Questions on Huffman Encoding - GeeksforGeeks

Huffman Encoding is an important topic from GATE point of view and different types of questions are asked from this topic. Before understanding this article, you should have basic idea about Huffman encoding.

These are the types of questions asked in GATE based on Huffman Encoding.

Type 1. Conceptual questions based on Huffman Encoding –
Here are the few key points based on Huffman Encoding:

- It is a lossless data compressing technique generating variable length codes for different symbols.
- It is based on greedy approach which considers frequency/probability of alphabets for generating codes.
- It has complexity of nlogn where n is the number of unique characters.
- The length of the code for a character is inversely proportional to frequency of its occurrence.
- No code is prefix of another code due to which a sequence of code can be unambiguously decoded to characters.

Que – 1. Which of the following is true about Huffman Coding?
(A) Huffman coding may become lossy in some cases
(B) Huffman Codes may not be optimal lossless codes in some cases
(C) In Huffman coding, no code is prefix of any other code.
(D) All of the above

Solution: As discussed, Huffman encoding is a lossless compression technique. Therefore, option (A) and (B) are false. Option (C) is true as this is the basis of decoding of message from given code.

Type 2. To find number of bits for encoding a given message –
To solve this type of questions:

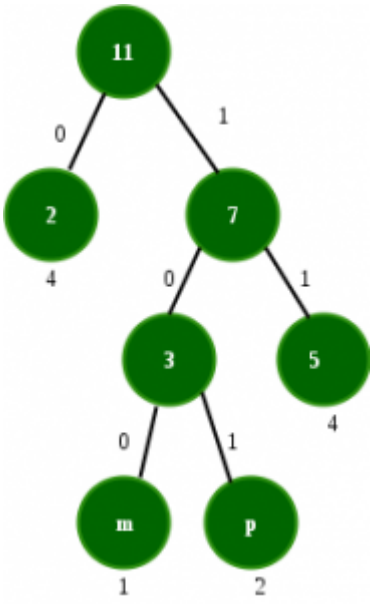
- First calculate frequency of characters if not given
- Generate Huffman Tree
- Calculate number of bits using frequency of characters and number of bits required to represent those characters.

Que – 2. How many bits may be required for encoding the message 'mississippi'?

Solution: Following is the frequency table of characters in 'mississippi' in non-decreasing order of frequency:

Character	Frequency
m	1
p	2
s	4
i	4

The generated Huffman tree is:



Following are the codes:

Character	Frequency	Code	Code Length
m	1	100	3
p	2	101	3
s	4	11	2
i	4	0	1

Total number of bits
= freq(m) * codelength(m) + freq(p) * code_length(p) + freq(s) * code_length(s) + freq(i) * code length(i)
= 1*3 + 2*3 + 4*2 + 4*1 = 21

Also, average bits per character can be found as:
Total number of bits required / total number of characters = 21/11 = 1.909

Type 3. Decoding from code to message –
To solve this type of question:

- Generate codes for each character using Huffman tree (if not given)
- Using prefix matching, replace the codes with characters.

Que – 3. The characters a to h have the set of frequencies based on the first 8 Fibonacci numbers as follows:

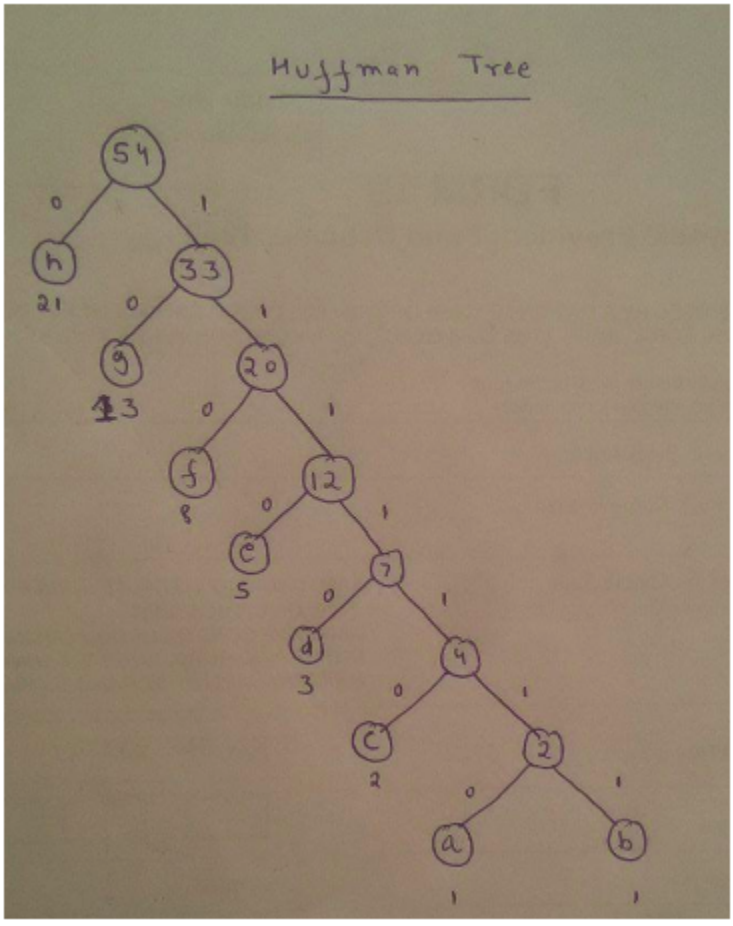
```
a : 1, b : 1, c : 2, d : 3, e : 5, f : 8, g : 13, h : 21
```

A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code?

```
110111100111010
```

(A) fdheg
(B) ecgdf
(C) dchfg
(D) fehdg

Solution: Using frequencies given in the question, huffman tree can be generated as:



Following are the codes:

Character	Code
a	1111110
b	1111111
c	111110
d	11110
e	1110
f	110
g	10
h	0

Using prefix matching, given string can be decomposed as

```
110 11110 0 1110 10
f   d     h  e   g
```

Type 4. Number of bits saved using Huffman encoding –

Que – 4. 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

Note that 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?

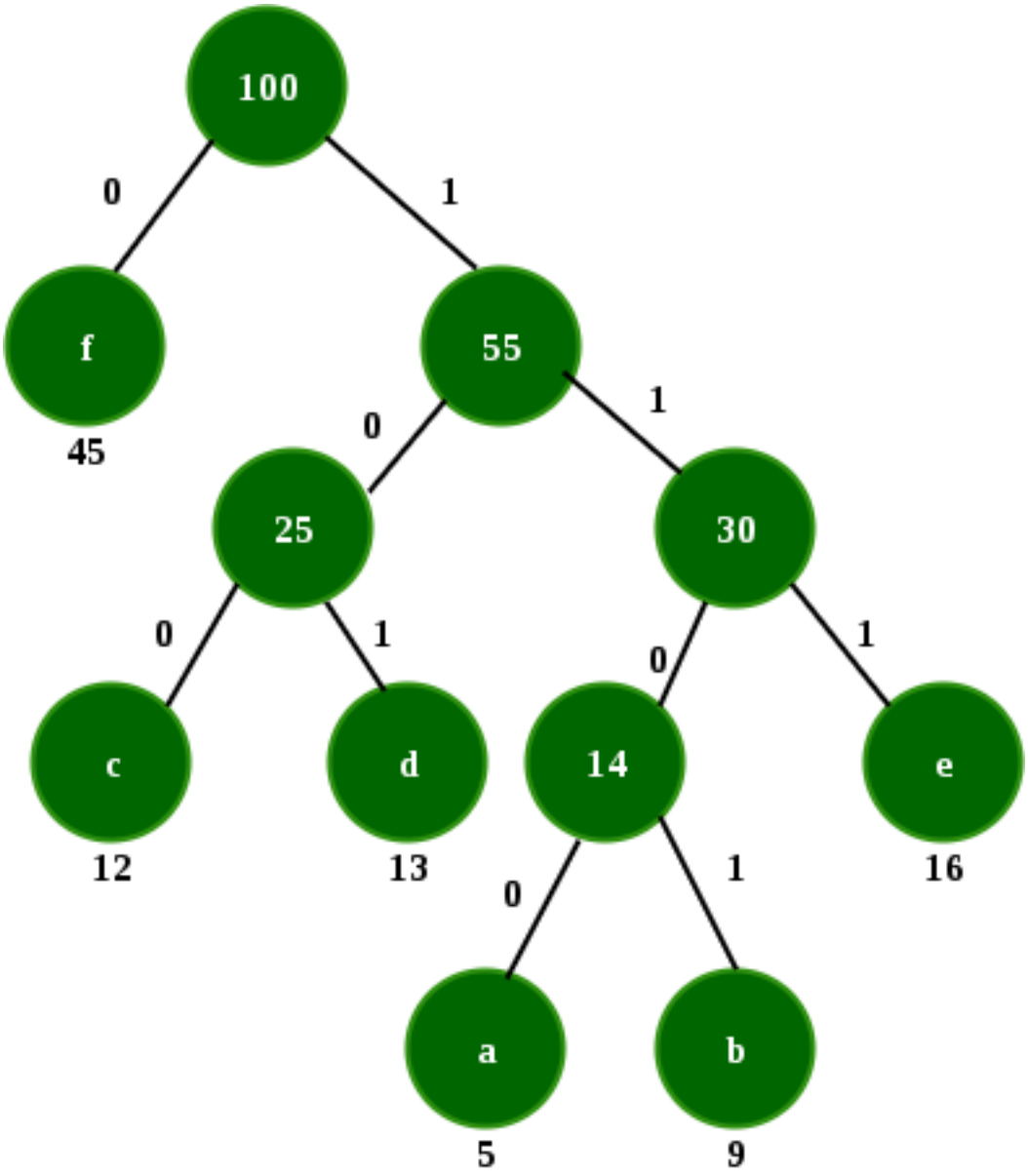
(A) 224
(B) 800
(C) 576
(D) 324

Solutions: Finding number of bits without using Huffman,
Total number of characters = sum of frequencies = 100
size of 1 character = 1byte = 8 bits
Total number of bits = 8*100 = 800

Using Huffman Encoding, Total number of bits needed can be calculated as:

5*4 + 9*4 + 12*3 + 13*3 + 16*3 + 45* 1 = 224

Bits saved = 800-224 = 576.



Source

https://www.geeksforgeeks.org/practice-questions-on-huffman-encoding/

Chapter 112

Prim's Algorithm (Simple Implementation for Adjacency Matrix Representation)

Prim's Algorithm (Simple Implementation for Adjacency Matrix Representation) - GeeksforGeeks

We have discussed Prim's algorithm and its implementation for adjacency matrix representation of graphs.
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. In every iteration, we consider the minimum weight edge among the edges that connect the two sets.

The implementation discussed in previous post uses two arrays to find minimum weight edge that connects the two sets. Here we use one inMST[V]. The value of MST[i] is going to be true if vertex i is included in the MST. In every pass, we consider only those edges such that one vertex of the edge is included in MST and other is not. After we pick an edge, we mark both vertices as included in MST.

```
 // A simple C++ implementation to find minimum
// spanning tree for adjacency representation.
#include <bits/stdc++.h>
using namespace std;
#define V 5

// Returns true if edge u-v is a valid edge to be
// include in MST. An edge is valid if one end is
// already included in MST and other is not in MST.
bool isValidEdge(int u, int v, vector<bool> inMST)
{
```

```
   if (u == v)
       return false;
   if (inMST[u] == false && inMST[v] == false)
        return false;
   else if (inMST[u] == true && inMST[v] == true)
        return false;
   return true;
}

void primMST(int cost[][V])
{
    vector<bool> inMST(V, false);

    // Include first vertex in MST
    inMST[0] = true;

    // Keep adding edges while number of included
    // edges does not become V-1.
    int edge_count = 0, mincost = 0;
    while (edge_count < V - 1) {

        // Find minimum weight valid edge.
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (cost[i][j] < min) {
                    if (isValidEdge(i, j, inMST)) {
                        min = cost[i][j];
                        a = i;
                        b = j;
                    }
                }
            }
        }
        if (a != -1 && b != -1) {
            printf("Edge %d:(%d, %d) cost: %d \n",
                        edge_count++, a, b, min);
            mincost = mincost + min;
            inMST[b] = inMST[a] = true;
        }
    }
    printf("\n Minimum cost= %d \n", mincost);
}

// 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 cost[][V] = {
        { INT_MAX, 2, INT_MAX, 6, INT_MAX },
        { 2, INT_MAX, 3, 8, 5 },
        { INT_MAX, 3, INT_MAX, INT_MAX, 7 },
        { 6, 8, INT_MAX, INT_MAX, 9 },
        { INT_MAX, 5, 7, 9, INT_MAX },
    };

    // Print the solution
    primMST(cost);

    return 0;
}
```

Output:

```
Edge 0:(0, 1) cost: 2
Edge 1:(1, 2) cost: 3
Edge 2:(1, 4) cost: 5
Edge 3:(0, 3) cost: 6

 Minimum cost= 16
```

Time Complexity : $O(V^3)$

Note that time complexity of previous approach that uses adjacency matrix is $O(V^2)$ and time complexity of the adjacency list representation implementation is $O((E+V)LogV)$.

Source

https://www.geeksforgeeks.org/prims-algorithm-simple-implementation-for-adjacency-matrix-representation/

Chapter 113

Prim's MST for Adjacency List Representation Greedy Algo-6

Prim's MST for Adjacency List Representation Greedy Algo-6 - GeeksforGeeks

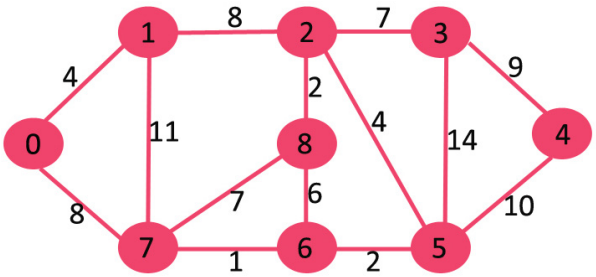
We recommend to read following two posts as a prerequisite of this post.

1. Greedy Algorithms Set 5 (Prim's Minimum Spanning Tree (MST))
2. Graph and its representations

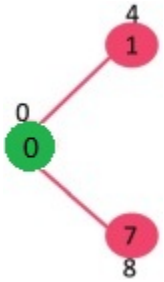
We have discussed Prim's algorithm and its implementation for adjacency matrix representation of graphs. The time complexity for the matrix representation is O(V^2). In this post, O(ELogV) algorithm for adjacency list representation is discussed.
As discussed in the previous post, in Prim's algorithm, 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 BFSand 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(LogV) 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.

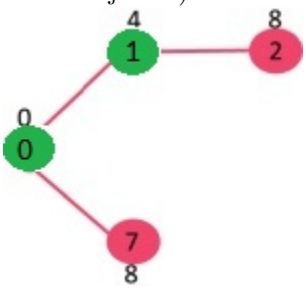
Let us understand the above algorithm with the following example:



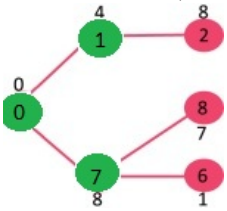
Initially, key value of first vertex is 0 and INF (infinite) for all other vertices. So vertex 0 is extracted from Min Heap and key values of vertices adjacent to 0 (1 and 7) are updated. Min Heap contains all vertices except vertex 0.
The vertices in green color are the vertices included in MST.



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.

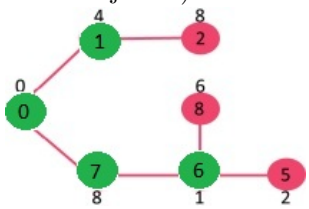


Since key value of vertex 7 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 7 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 7 to the adjacent). Min Heap contains all vertices except vertex 0, 1 and 7.

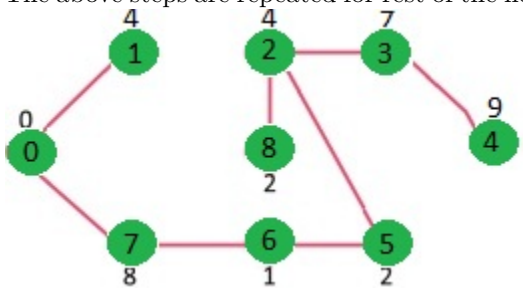


Since key value of vertex 6 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 6 are updated (Key is updated if the a

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 / C++ program for Prim's MST for adjacency list representation of graph

#include <limits.h>
#include <stdio.h>
#include <stdlib.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;
}
```

Java

```
 // Java program for Prim's MST for
// adjacency list representation of graph
import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Comparator;

public class prims {
    class node1 {

        // Stores destination vertex in adjacency list
        int dest;

        // Stores weight of a vertex in adjacency list
        int weight;

        // Constructor
        node1(int a, int b)
        {
            dest = a;
            weight = b;
```

```
    }
}
static class Graph {

    // Number of vertices in the graph
    int V;

    // List of adjacent nodes of a given vertex
    LinkedList<node1>[] adj;

    // Constructor
    Graph(int e)
    {
        V = e;
        adj = new LinkedList[V];
        for (int o = 0; o < V; o++)
            adj[o] = new LinkedList<>();
    }
}

// class to represent a node in PriorityQueue
// Stores a vertex and its corresponding
// key value
class node {
    int vertex;
    int key;
}

// Comparator class created for PriorityQueue
// returns 1 if node0.key > node1.key
// returns 0 if node0.key < node1.key and
// returns -1 otherwise
class comparator implements Comparator<node> {

    @Override
    public int compare(node node0, node node1)
    {
        return node0.key - node1.key;
    }
}

// method to add an edge
// between two vertices
void addEdge(Graph graph, int src, int dest, int weight)
{

    node1 node0 = new node1(dest, weight);
    node1 node = new node1(src, weight);
```

```
        graph.adj[src].addLast(node0);
        graph.adj[dest].addLast(node);
    }

    // method used to find the mst
    void prims_mst(Graph graph)
    {

        // Whether a vertex is in PriorityQueue or not
        Boolean[] mstset = new Boolean[graph.V];
        node[] e = new node[graph.V];

        // Stores the parents of a vertex
        int[] parent = new int[graph.V];

        for (int o = 0; o < graph.V; o++)
            e[o] = new node();

        for (int o = 0; o < graph.V; o++) {

            // Initialize to false
            mstset[o] = false;

            // Initialize key values to infinity
            e[o].key = Integer.MAX_VALUE;
            e[o].vertex = o;
            parent[o] = -1;
        }

        // Include the source vertex in mstset
        mstset[0] = true;

        // Set key value to 0
        // so that it is extracted first
        // out of PriorityQueue
        e[0].key = 0;

        // PriorityQueue
        PriorityQueue<node> queue = new PriorityQueue<>(graph.V, new comparator());

        for (int o = 0; o < graph.V; o++)
            queue.add(e[o]);

        // Loops until the PriorityQueue is not empty
        while (!queue.isEmpty()) {

            // Extracts a node with min key value
            node node0 = queue.poll();
```

```
            // Include that node into mstset
            mstset[node0.vertex] = true;

            // For all adjacent vertex of the extracted vertex V
            for (node1 iterator : graph.adj[node0.vertex]) {

                // If V is in PriorityQueue
                if (mstset[iterator.dest] == false) {
                    // If the key value of the adjacent vertex is
                    // more than the extracted key
                    // update the key value of adjacent vertex
                    // to update first remove and add the updated vertex
                    if (e[iterator.dest].key > iterator.weight) {
                        queue.remove(e[iterator.dest]);
                        e[iterator.dest].key = iterator.weight;
                        queue.add(e[iterator.dest]);
                        parent[iterator.dest] = node0.vertex;
                    }
                }
            }
        }

        // Prints the vertex pair of mst
        for (int o = 1; o < graph.V; o++)
            System.out.println(parent[o] + " "
                               + "-"
                               + " " + o);
    }

    public static void main(String[] args)
    {
        int V = 9;

        Graph graph = new Graph(V);

        prims e = new prims();

        e.addEdge(graph, 0, 1, 4);
        e.addEdge(graph, 0, 7, 8);
        e.addEdge(graph, 1, 2, 8);
        e.addEdge(graph, 1, 7, 11);
        e.addEdge(graph, 2, 3, 7);
        e.addEdge(graph, 2, 8, 2);
        e.addEdge(graph, 2, 5, 4);
        e.addEdge(graph, 3, 4, 9);
        e.addEdge(graph, 3, 5, 14);
        e.addEdge(graph, 4, 5, 10);
```

```
        e.addEdge(graph, 5, 6, 2);
        e.addEdge(graph, 6, 7, 1);
        e.addEdge(graph, 6, 8, 6);
        e.addEdge(graph, 7, 8, 7);

        // Method invoked
        e.prims_mst(graph);
    }
}
// This code is contributed by Vikash Kumar Dubey
```

Python

```
 # A Python program for Prims's MST for
# adjacency list representation of graph

from collections import defaultdict
import sys

class Heap():

    def __init__(self):
        self.array = []
        self.size = 0
        self.pos = []

    def newMinHeapNode(self, v, dist):
        minHeapNode = [v, dist]
        return minHeapNode

    # A utility function to swap two nodes of
    # min heap. Needed for min heapify
    def swapMinHeapNode(self, a, b):
        t = self.array[a]
        self.array[a] = self.array[b]
        self.array[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()
    def minHeapify(self, idx):
        smallest = idx
        left = 2 * idx + 1
        right = 2 * idx + 2

        if left < self.size and self.array[left][1] < \
                                self.array[smallest][1]:
```

```
            smallest = left

        if right < self.size and self.array[right][1] < \
                                self.array[smallest][1]:
            smallest = right

        # The nodes to be swapped in min heap
        # if idx is not smallest
        if smallest != idx:

            # Swap positions
            self.pos[ self.array[smallest][0] ] = idx
            self.pos[ self.array[idx][0] ] = smallest

            # Swap nodes
            self.swapMinHeapNode(smallest, idx)

            self.minHeapify(smallest)

    # Standard function to extract minimum node from heap
    def extractMin(self):

        # Return NULL wif heap is empty
        if self.isEmpty() == True:
            return

        # Store the root node
        root = self.array[0]

        # Replace root node with last node
        lastNode = self.array[self.size - 1]
        self.array[0] = lastNode

        # Update position of last node
        self.pos[lastNode[0]] = 0
        self.pos[root[0]] = self.size - 1

        # Reduce heap size and heapify root
        self.size -= 1
        self.minHeapify(0)

        return root

    def isEmpty(self):
        return True if self.size == 0 else False

    def decreaseKey(self, v, dist):
```

```
        # Get the index of v in  heap array

        i = self.pos[v]

        # Get the node and update its dist value
        self.array[i][1] = dist

        # Travel up while the complete tree is not
        # hepified. This is a O(Logn) loop
        while i > 0 and self.array[i][1] < \
                    self.array[(i - 1) / 2][1]:

            # Swap this node with its parent
            self.pos[ self.array[i][0] ] = (i-1)/2
            self.pos[ self.array[(i-1)/2][0] ] = i
            self.swapMinHeapNode(i, (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
    def isInMinHeap(self, v):

        if self.pos[v] < self.size:
            return True
        return False


def printArr(parent, n):
    for i in range(1, n):
        print "% d - % d" % (parent[i], i)


class Graph():

    def __init__(self, V):
        self.V = V
        self.graph = defaultdict(list)

    # Adds an edge to an undirected graph
    def addEdge(self, src, dest, 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. The first element of
        # the node has the destination and the second
        # elements has the weight
```

```
        newNode = [dest, weight]
        self.graph[src].insert(0, newNode)

        # Since graph is undirected, add an edge from
        # dest to src also
        newNode = [src, weight]
        self.graph[dest].insert(0, newNode)

    # The main function that prints the Minimum
    # Spanning Tree(MST) using the Prim's Algorithm.
    # It is a O(ELogV) function
    def PrimMST(self):
        # Get the number of vertices in graph
        V = self.V

        # key values used to pick minimum weight edge in cut
        key = []

        # List to store contructed MST
        parent = []

        # minHeap represents set E
        minHeap = Heap()

        # Initialize min heap with all vertices. Key values of all
        # vertices (except the 0th vertex) is is initially infinite
        for v in range(V):
            parent.append(-1)
            key.append(sys.maxint)
            minHeap.array.append( minHeap.newMinHeapNode(v, key[v]) )
            minHeap.pos.append(v)

        # Make key value of 0th vertex as 0 so
        # that it is extracted first
        minHeap.pos[0] = 0
        key[0] = 0
        minHeap.decreaseKey(0, key[0])

        # Initially size of min heap is equal to V
        minHeap.size = V;

        # In the following loop, min heap contains all nodes
        # not yet added in the MST.
        while minHeap.isEmpty() == False:

            # Extract the vertex with minimum distance value
            newHeapNode = minHeap.extractMin()
            u = newHeapNode[0]
```

```
            # Traverse through all adjacent vertices of u
            # (the extracted vertex) and update their
            # distance values
            for pCrawl in self.graph[u]:

                v = pCrawl[0]

                # If shortest distance to v is not finalized
                # yet, and distance to v through u is less than
                # its previously calculated distance
                if minHeap.isInMinHeap(v) and pCrawl[1] < key[v]:
                    key[v] = pCrawl[1]
                    parent[v] = u

                    # update distance value in min heap also
                    minHeap.decreaseKey(v, key[v])

        printArr(parent, V)


# Driver program to test the above functions
graph = Graph(9)
graph.addEdge(0, 1, 4)
graph.addEdge(0, 7, 8)
graph.addEdge(1, 2, 8)
graph.addEdge(1, 7, 11)
graph.addEdge(2, 3, 7)
graph.addEdge(2, 8, 2)
graph.addEdge(2, 5, 4)
graph.addEdge(3, 4, 9)
graph.addEdge(3, 5, 14)
graph.addEdge(4, 5, 10)
graph.addEdge(5, 6, 2)
graph.addEdge(6, 7, 1)
graph.addEdge(6, 8, 6)
graph.addEdge(7, 8, 7)
graph.PrimMST()

# This code is contributed by Divyanshu Mehta
```

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(LogV) time. So overall time complexity is O(E+V)*O(LogV) which is O((E+V)*LogV) = O(ELogV) (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.

Improved By : VikashDubey, Sumon Nath

Source

https://www.geeksforgeeks.org/prims-mst-for-adjacency-list-representation-greedy-algo-6/

Chapter 114

Prim's Minimum Spanning Tree (MST) Greedy Algo-5

Prim's Minimum Spanning Tree (MST) Greedy Algo-5 - GeeksforGeeks

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 verices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).*

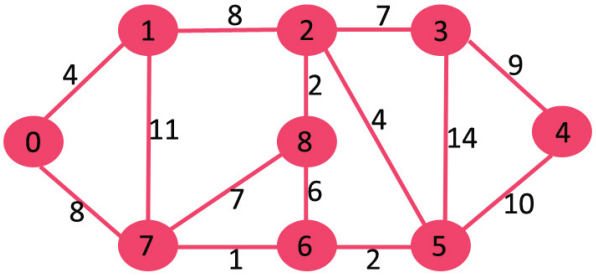
How does Prim's Algorithm Work? The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning* Tree. And they must be connected with the minimum weight edge to make it a *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

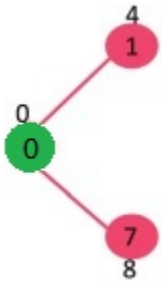
The idea of using key values is to pick the minimum weight edge from cut. The key values

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.

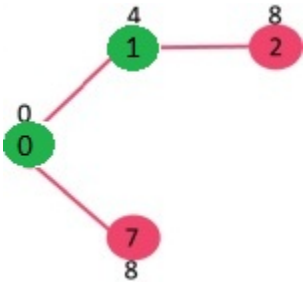
Let us understand with the following example:



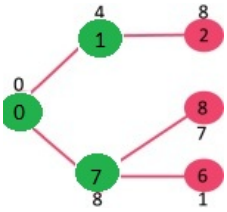
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet*, update key values 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.



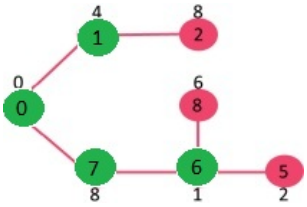
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



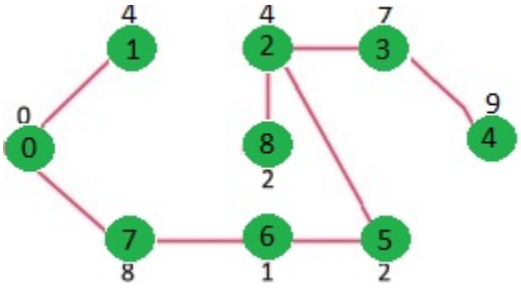
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (7 and 1 respectively).



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.



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.

C/C++

```
 // 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>
#include<stdbool.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 \tWeight\n");
for (int i = 1; i < V; i++)
    printf("%d - %d \t%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])
{
    // Array to store constructed MST
    int parent[V];
    // Key values used to pick minimum weight edge in cut
    int key[V];
    // To represent set of vertices not yet included in MST
    bool mstSet[V];

    // 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.
    // Make key 0 so that this vertex is picked as first vertex.
    key[0] = 0;
    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 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] && 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;
}
```

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 \tWeight");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i]+" - "+ i+"\t"+
                               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}};

        // Print the solution
        t.primMST(graph);
    }
}
// This code is contributed by Aakash Hasija
```

Python

```
 # A Python program for Prim's Minimum Spanning Tree (MST) algorithm.
# The program is for adjacency matrix representation of the graph

import sys # Library for INT_MAX

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                    for row in range(vertices)]

    # A utility function to print the constructed MST stored in parent[]
    def printMST(self, parent):
        print "Edge \tWeight"
        for i in range(1,self.V):
            print parent[i],"-",i,"\t",self.graph[i][ parent[i] ]

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):

        # Initilaize min value
        min = sys.maxint
```

```
        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index

    # Function to construct and print MST for a graph
    # represented using adjacency matrix representation
    def primMST(self):

        #Key values used to pick minimum weight edge in cut
        key = [sys.maxint] * self.V
        parent = [None] * self.V # Array to store constructed MST
        # Make key 0 so that this vertex is picked as first vertex
        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1 # First node is always the root of

        for cout in range(self.V):

            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minKey(key, mstSet)

            # Put the minimum distance vertex in
            # the shortest path tree
            mstSet[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shotest path tree
            for v in range(self.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 self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u

        self.printMST(parent)

g = Graph(5)
g.graph = [ [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]]

g.primMST();

# Contributed by Divyanshu Mehta
```

C#

```
 // A C# program for Prim's Minimum
// Spanning Tree (MST) algorithm.
// The program is for adjacency
// matrix representation of the graph
using System;
class MST {

    // Number of vertices in the graph
    static int V = 5;

    // A utility function to find
    // the vertex with minimum key
    // value, from the set of vertices
    // not yet included in MST
    static int minKey(int []key, bool []mstSet)
    {

        // Initialize min value
        int min = int.MaxValue, 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[]
    static void printMST(int []parent, int n, int [,]graph)
    {
        Console.WriteLine("Edge \tWeight");
        for (int i = 1; i < V; i++)
```

```
        Console.WriteLine(parent[i]+" - "+ i+"\t"+
                          graph[i,parent[i]]);
}

// Function to construct and
// print MST for a graph represented
// using adjacency matrix representation
static 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
    bool []mstSet = new bool[V];

    // Initialize all keys
    // as INFINITE
    for (int i = 0; i < V; i++)
    {
        key[i] = int.MaxValue;
        mstSet[i] = false;
    }

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is
    // picked as first vertex
    // First node is always root of MST
    key[0] = 0;
    parent[0] = -1;

    // 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);
}

// Driver Code
public static void Main ()
{

    /* Let us create the following graph
    2 3
    (0)--(1)--(2)
    | / \ |
    6| 8/ \5 |7
    | / \ |
    (3)-------(4)
        9 */

    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}};

    // Print the solution
    primMST(graph);
}
```

```
}

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

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.

Improved By : vt_m, AnkurKarmakar

Source

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Chapter 115

Print a closest string that does not contain adjacent duplicates

Print a closest string that does not contain adjacent duplicates - GeeksforGeeks

Given a string S, change the smallest number of letters in S such that all adjacent characters are different. Print the resultant string.

Examples :

```
Input : S = "aab"
Output: acb
Explanation :
Loop will start for i-th character, which
is second ‘a’. It’s cannot be ‘b’ since it
matches with third char. So output should
be ‘acb’.

Input : S = "geeksforgeeks"
Output: geaksforgeaks
Explanation :
Resultant string, after making minimal
changes. S = "geaksforgeaks". We made two
changes, which is the optimal solution here.
```

We can solve this problem using greedy approach. Let us consider a segment of length k of consecutive identical characters. We have to make at least [K/2] changes in the segment, to make that there are no identical characters in a row. We can also change the second, fourth etc.. characters of the string that is it should not be equal to the letter on the left side and the letter to the right side.

Traverse the string from starting index (i = 1) and if any two adjacent letters(i & i-1) are equal then initialize (i)th character with 'a' and start another loop to make (i)th character different from the left and right letters.

Below is the implementation of above approach :
C++

```
 // C++ program to print a string with no adjacent
// duplicates by doing  minimum changes to original
// string
#include <bits/stdc++.h>
using namespace std;

// Function to print simple string
string noAdjacentDup(string s)
{
    int n = s.length();
    for (int i = 1; i < n; i++)
    {
        // If any two adjacent characters are equal
        if (s[i] == s[i - 1])
        {
            s[i] = 'a'; // Initialize it to 'a'

            // Traverse the loop until it is different
            // from the left and right letter.
            while (s[i] == s[i - 1] ||
                (i + 1 < n && s[i] == s[i + 1]))
                s[i]++;

            i++;
        }
    }
    return s;
}

// Driver Function
int main()
{
    string s = "geeksforgeeks";
    cout << noAdjacentDup(s);
    return 0;
}
```

Java

```
 // Java program to print a string with
// no adjacent duplicates by doing
```

```
// minimum changes to original string
import java.util.*;
import java.lang.*;

public class GfG{

    // Function to print simple string
    public static String noAdjacentDup(String s1)
    {
        int n = s1.length();
        char[] s = s1.toCharArray();
        for (int i = 1; i < n; i++)
        {
            // If any two adjacent
            // characters are equal
            if (s[i] == s[i - 1])
            {
                // Initialize it to 'a'
                s[i] = 'a';

                // Traverse the loop until it
                // is different from the left
                // and right letter.
                while (s[i] == s[i - 1] ||
                    (i + 1 < n && s[i] == s[i + 1]))
                    s[i]++;

                i++;
            }
        }
        return (new String(s));
    }

    // Driver function
    public static void main(String argc[]){

        String s = "geeksforgeeks";
        System.out.println(noAdjacentDup(s));

    }

}

/* This code is contributed by Sagar Shukla */
```

Python3

```
 # Python program to print a string with
```

```
# no adjacent duplicates by doing minimum
# changes to original string

# Function to print simple string
def noAdjacentDup(s):

    n = len(s)
    for i in range(1, n):

        # If any two adjacent characters are equal
        if (s[i] == s[i - 1]):

            s[i] = "a" # Initialize it to 'a'

            # Traverse the loop until it is different
            # from the left and right letter.
            while (s[i] == s[i - 1] or
                (i + 1 < n and s[i] == s[i + 1])):
                s[i] += 1

            i += 1

    return s

# Driver Function
s = list("geeksforgeeks")
print("".join(noAdjacentDup(s)))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to print a string with
// no adjacent duplicates by doing
// minimum changes to original string
using System;

class GfG{

    // Function to print simple string
    public static String noAdjacentDup(String s1)
    {
        int n = s1.Length;

        char[] s = s1.ToCharArray();
        for (int i = 1; i < n; i++)
        {
            // If any two adjacent
```

```
            // characters are equal
            if (s[i] == s[i - 1])
            {
                // Initialize it to 'a'
                s[i] = 'a';

                // Traverse the loop until it
                // is different from the left
                // and right letter.
                while (s[i] == s[i - 1] ||
                      (i + 1 < n && s[i] == s[i + 1]))
                 s[i]++;

                i++;
            }
        }
        return (new String(s));
    }

    // Driver function
    public static void Main(String[] argc)
    {
        String s = "geeksforgeeks";

        // Function calling
        Console.Write(noAdjacentDup(s));
    }
}

/* This code is contributed by parashar */
```

PHP

```
 <?php
// PHP program to print a
// string with no adjacent
// duplicates by doing minimum
// changes to original string

// Function to print
// simple string
function noAdjacentDup($s)
{
    $n = strlen($s);
    for ($i = 1; $i < $n; $i++)
    {
        // If any two adjacent
        // characters are equal
```

```
        if ($s[$i] == $s[$i - 1])
        {
            // Initialize it to 'a'
            $s[$i] = 'a';

            // Traverse the loop
            // until it is different
            // from the left and
            // right letter.
            while ($s[$i] == $s[$i - 1] ||
                  ($i + 1 < $n &&
                   $s[$i] == $s[$i + 1]))
                $s[$i]++;

            $i++;
        }
    }
    return $s;
}

// Driver Code
$s = "geeksforgeeks";
echo (noAdjacentDup($s));

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
geaksforgeaks
```

Improved By : parashar, manishshaw1

Source

https://www.geeksforgeeks.org/print-a-closest-string-that-does-not-contain-adjacent-duplicates/

Chapter 116

Problem Solving for Minimum Spanning Trees (Kruskal's and Prim's)

Problem Solving for Minimum Spanning Trees (Kruskal's and Prim's) - GeeksforGeeks

Minimum spanning Tree (MST) is an important topic for GATE. Therefore, we will discuss how to solve different types of questions based on MST. Before understanding this article, you should understand basics of MST and their algorithms (Kruskal's algorithm and Prim's algorithm).

Type 1. Conceptual questions based on MST –
There are some important properties of MST on the basis of which conceptual questions can be asked as:

- The number of edges in MST with n nodes is (n-1).
- The weight of MST of a graph is always unique. However there may be different ways to get this weight (if there edges with same weights).
- The weight of MST is sum of weights of edges in MST.
- Maximum path length between two vertices is (n-1) for MST with n vertices.
- There exists only one path from one vertex to another in MST.
- Removal of any edge from MST disconnects the graph.
- For a graph having edges with distinct weights, MST is unique.

Que – 1. Let G be an undirected connected graph with distinct edge weight. Let emax be the edge with maximum weight and emin the edge with minimum weight. Which of the following statements is false? (GATE CS 2000)

(A) Every minimum spanning tree of G must contain emin.
(B) If emax is in a minimum spanning tree, then its removal must disconnect G
(C) No minimum spanning tree contains emax
(D) G has a unique minimum spanning tree

Solution: As edge weights are unique, there will be only one edge emin and that will be added to MST, therefore option (A) is always true.
As spanning tree has minimum number of edges, removal of any edge will disconnect the graph. Therefore, option (B) is also true.
As all edge weights are distinct, G will have a unique minimum spanning tree. So, option (D) is correct.
Option C is false as emax can be part of MST if other edges with lesser weights are creating cycle and number of edges before adding emax is less than (n-1).

Type 2. How to find the weight of minimum spanning tree given the graph –
This is the simplest type of question based on MST. To solve this using kruskal's algorithm,

- Arrange the edges in non-decreasing order of weights.
- Add edges one by one if they don't create cycle until we get n-1 number of edges where n are number of nodes in the graph.

Que – 2. Consider a complete undirected graph with vertex set {0, 1, 2, 3, 4}. Entry Wij in the matrix W below is the weight of the edge {i, j}. What is the minimum possible weight of a spanning tree T in this graph such that vertex 1 is a leaf node in the tree T? (GATE CS 2010)

0	1	8	1	4
1	0	12	4	9
8	12	0	7	3
1	4	7	0	2
4	9	3	2	0

(A) 7
(B) 8
(C) 9
(D) 10

Solution: In the adjacency matrix of the graph with 5 vertices (v1 to v5), the edges arranged in non-decreasing order are:

```
(v1,v2), (v1,v4), (v4,v5), (v3,v5), (v1,v5),
(v2,v4), (v3,v4), (v1,v3), (v2,v5), (v2,v3)
```

As it is given, vertex v1 is a leaf node, it should have only one edge incident to it. Therefore, we will consider it in the end. Considering vertices v2 to v5, edges in non decreasing order are:

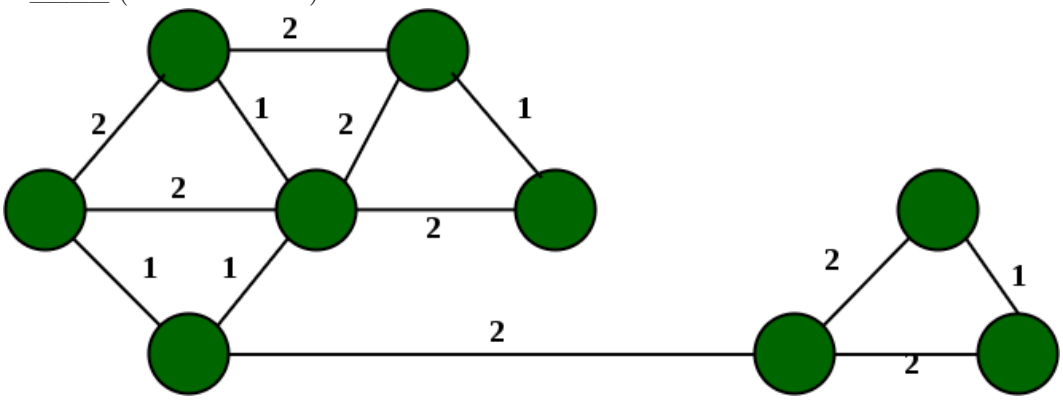
```
(v4,v5), (v3,v5), (v2,v4), (v3,v4), (v2,v5), (v2,v3)
```

Adding first three edges (v4,v5), (v3,v5), (v2,v4), no cycle is created. Also, we can connect v1 to v2 using edge (v1,v2). The total weight is sum of weight of these 4 edges which is 10.

Type 3. How many minimum spanning trees are possible using Kruskal's algorithm for a given graph –

- If all edges weight are distinct, minimum spanning tree is unique.
- If two edges have same weight, then we have to consider both possibilities and find possible minimum spanning trees.

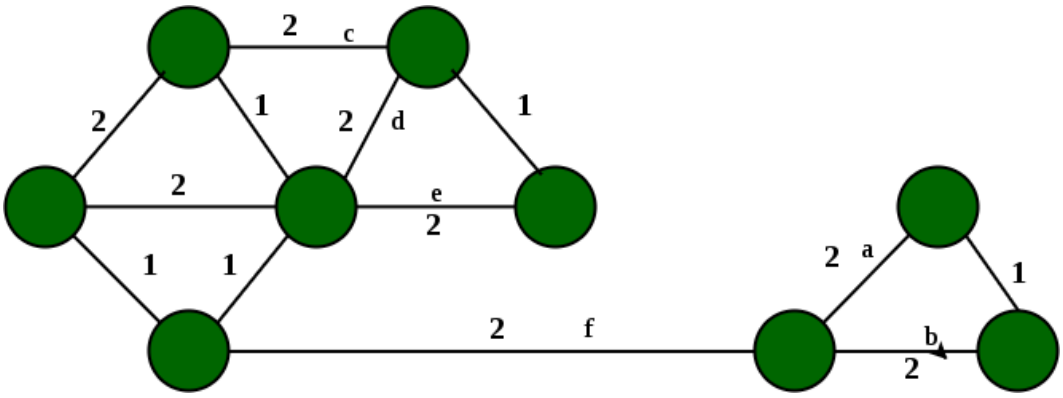
Que – 3. The number of distinct minimum spanning trees for the weighted graph below is _____ (GATE-CS-2014)



(A) 4
(B) 5
(C) 6
(D) 7

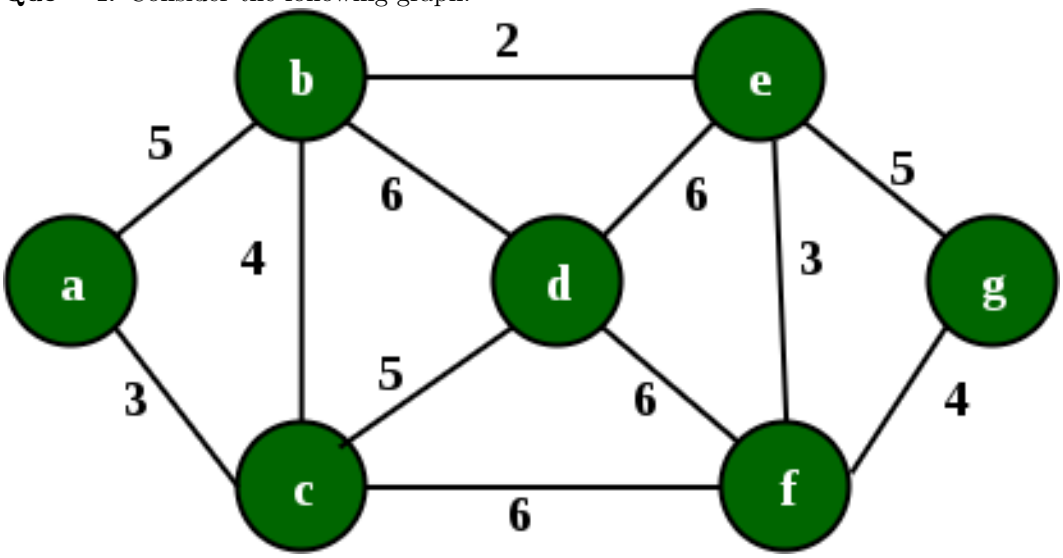
Solution: There are 5 edges with weight 1 and adding them all in MST does not create cycle.
As the graph has 9 vertices, therefore we require total 8 edges out of which 5 has been added. Out of remaining 3, one edge is fixed represented by f.

For remaining 2 edges, one is to be chosen from c or d or e and another one is to be chosen from a or b. Remaining black ones will always create cycle so they are not considered. So, possible MST are 3*2 = 6.



Type 4. Out of given sequences, which one is not the sequence of edges added to the MST using Kruskal's algorithm –
To solve this type of questions, try to find out the sequence of edges which can be produced by Kruskal. The sequence which does not match will be the answer.

Que – 4. Consider the following graph:



Which one of the following is NOT the sequence of edges added to the minimum spanning tree using Kruskal's algorithm? (GATE-CS-2009)
(A) (b,e), (e,f), (a,c), (b,c), (f,g), (c,d)
(B) (b,e), (e,f), (a,c), (f,g), (b,c), (c,d)
(C) (b,e), (a,c), (e,f), (b,c), (f,g), (c,d)
(D) (b,e), (e,f), (b,c), (a,c), (f,g), (c,d)

Solution: Kruskal algorithms adds the edges in non-decreasing order of their weights, therefore, we first sort the edges in non-decreasing order of weight as:

```
(b,e), (e,f), (a,c), (b,c), (f,g), (a,b), (e,g), (c,d), (b,d), (e,d), (d,f).
```

First it will add (b,e) in MST. Then, it will add (e,f) as well as (a,c) (either (e,f) followed by (a,c) or vice versa) because of both having same weight and adding both of them will not create cycle.
However, in option (D), (b,c) has been added to MST before adding (a,c). So it can't be the sequence produced by Kruskal's algorithm.

Source

https://www.geeksforgeeks.org/problem-solving-minimum-spanning-trees-kruskals-prims/

Chapter 117

Program for Best Fit algorithm in Memory Management

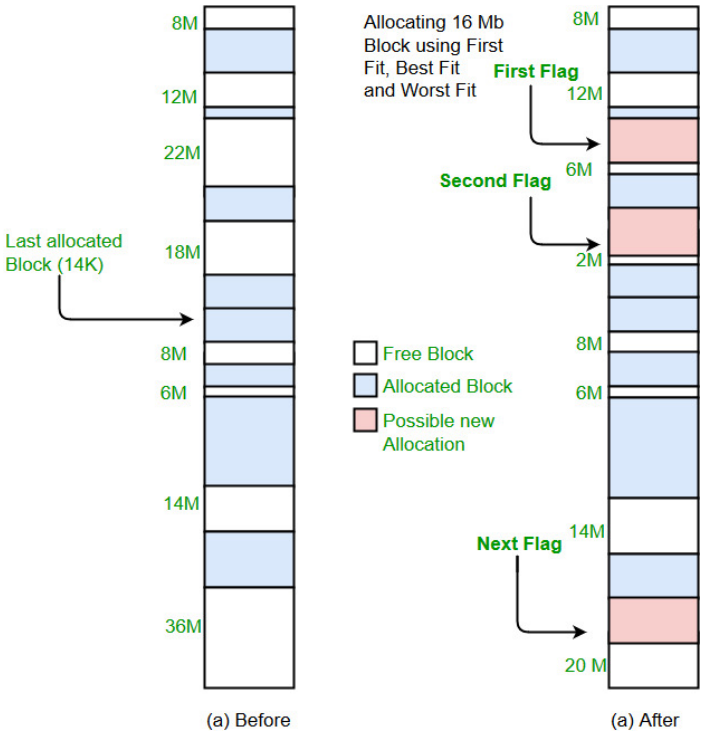
Program for Best Fit algorithm in Memory Management - GeeksforGeeks

Prerequisite : Partition allocation methods

Best fit allocates the process to a partition which is the smallest sufficient partition among the free available partitions.

Example:

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
 1        212        4
 2        417        2
 3        112        3
 4        426        5
```




```
Implementation:
1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the
   minimum block size that can be assigned to
   current process i.e., find min(bockSize[1],
   blockSize[2],.....blockSize[n]) >
   processSize[current], if found then assign
   it to the current process.
5- If not then leave that process and keep checking
   the further processes.
```

Below is implementation.

C/C++

```
 // C++ implementation of Best - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to blocks as per Best fit
// algorithm
void bestFit(int blockSize[], int m, int processSize[], int n)
```

```
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process
        int bestIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }

        // If we could find a block for current process
        if (bestIdx != -1)
        {
            // allocate block j to p[i] process
            allocation[i] = bestIdx;

            // Reduce available memory in this block.
            blockSize[bestIdx] -= processSize[i];
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
```

```
// Driver code
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);

    return 0 ;
}
```

Java

```
 // Java implementation of Best - Fit algorithm

public class GFG
{
    // Method to allocate memory to blocks as per Best fit
    // algorithm
    static void bestFit(int blockSize[], int m, int processSize[],
                                                        int n)
    {
        // Stores block id of the block allocated to a
        // process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

     // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i=0; i<n; i++)
        {
            // Find the best fit block for current process
            int bestIdx = -1;
            for (int j=0; j<m; j++)
            {
                if (blockSize[j] >= processSize[i])
                {
                    if (bestIdx == -1)
                        bestIdx = j;
                    else if (blockSize[bestIdx] > blockSize[j])
                        bestIdx = j;
                }
            }
```

```
            // If we could find a block for current process
            if (bestIdx != -1)
            {
                // allocate block j to p[i] process
                allocation[i] = bestIdx;

                // Reduce available memory in this block.
                blockSize[bestIdx] -= processSize[i];
            }
        }

        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            System.out.print("   " + (i+1) + "\t\t" + processSize[i] + "\t\t");
            if (allocation[i] != -1)
                System.out.print(allocation[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
    }

    // Driver Method
    public static void main(String[] args)
    {
         int blockSize[] = {100, 500, 200, 300, 600};
         int processSize[] = {212, 417, 112, 426};
         int m = blockSize.length;
         int n = processSize.length;

         bestFit(blockSize, m, processSize, n);
    }
}
```

C#

```
 // C# implementation of Best - Fit algorithm
using System;

public class GFG {

    // Method to allocate memory to blocks
    // as per Best fit
    // algorithm
    static void bestFit(int []blockSize, int m,
                        int []processSize, int n)
```

```
{

    // Stores block id of the block
    // allocated to a process
    int []allocation = new int[n];

    // Initially no block is assigned to
    // any process
    for (int i = 0; i < allocation.Length; i++)
        allocation[i] = -1;

    // pick each process and find suitable
    // blocks according to its size ad
    // assign to it
    for (int i = 0; i < n; i++)
    {

        // Find the best fit block for
        // current process
        int bestIdx = -1;
        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx]
                                > blockSize[j])
                    bestIdx = j;
            }
        }

        // If we could find a block for
        // current process
        if (bestIdx != -1)
        {

            // allocate block j to p[i]
            // process
            allocation[i] = bestIdx;

            // Reduce available memory in
            // this block.
            blockSize[bestIdx] -= processSize[i];
        }
    }

    Console.WriteLine("\nProcess No.\tProcess"
```

```
                            + " Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            Console.Write(" " + (i+1) + "\t\t"
                          + processSize[i] + "\t\t");

            if (allocation[i] != -1)
                Console.Write(allocation[i] + 1);
            else
                Console.Write("Not Allocated");

            Console.WriteLine();
        }
    }

    // Driver Method
    public static void Main()
    {
        int []blockSize = {100, 500, 200, 300, 600};
        int []processSize = {212, 417, 112, 426};
        int m = blockSize.Length;
        int n = processSize.Length;

        bestFit(blockSize, m, processSize, n);
    }
}

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

Output:

```
Process No.    Process Size    Block no.
 1        212        4
 2        417        2
 3        112        3
 4        426        5
```

Is Best-Fit really best?
Although, best fit minimizes the wastage space, it consumes a lot of processor time for searching the block which is close to required size. Also, Best-fit may perform poorer than other algorithms in some cases. For example, see below exercise.

Example: Consider the requests from processes in given order 300K, 25K, 125K and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.

Best Fit:
300K is allocated from block of size 350K. 50 is left in the block.
25K is allocated from the remaining 50K block. 25K is left in the block.
125K is allocated from 150 K block. 25K is left in this block also.
50K can't be allocated even if there is 25K + 25K space available.

First Fit:
300K request is allocated from 350K block, 50K is left out.
25K is be allocated from 150K block, 125K is left out.
Then 125K and 50K are allocated to remaining left out partitions.
So, first fit can handle requests.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/program-best-fit-algorithm-memory-management/

Chapter 118

Program for First Fit algorithm in Memory Management

Program for First Fit algorithm in Memory Management - GeeksforGeeks

Prerequisite : Partition Allocation Methods

In the first fit, the partition is allocated which is first sufficient from the top of Main Memory.

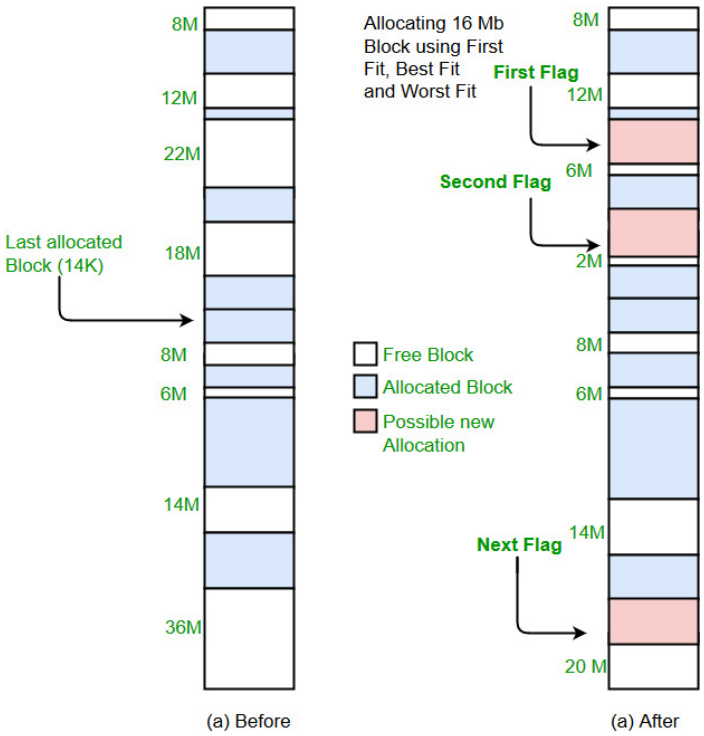
Example :

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
   1               212            2
   2               417            5
   3               112            2
   4               426        Not Allocated
```

- Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.
- It may have problems of not allowing processes to take space even if it was possible to allocate. Consider the above example, process number 4 (of size 426) does not get memory. However it was possible to allocate memory if we had allocated using best fit allocation [block number 4 (of size 300) to process 1, block number 2 to process 2, block number 3 to process 3 and block number 5 to process 4].

```
Implementation:
1- Input memory blocks with size and processes with size.
```

```
2- Initialize all memory blocks as free.
3- Start by picking each process and check if it can
   be assigned to current block.
4- If size-of-process <= size-of-block if yes then
   assign and check for next process.
5- If not then keep checking the further blocks.
```



Below is an implementation of above steps.

C++

```
 // C++ implementation of First - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to
// blocks as per First fit algorithm
void firstFit(int blockSize[], int m,
              int processSize[], int n)
{
    // Stores block id of the
    // block allocated to a process
    int allocation[n];

    // Initially no block is assigned to any process
```

```
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                // allocate block j to p[i] process
                allocation[i] = j;

                // Reduce available memory in this block.
                blockSize[j] -= processSize[i];

                break;
            }
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t"
             << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

// Driver code
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

Java

```
 // Java implementation of First - Fit algorithm

// Java implementation of First - Fit algorithm
class GFG
{
    // Method to allocate memory to
    // blocks as per First fit algorithm
    static void firstFit(int blockSize[], int m,
                         int processSize[], int n)
    {
        // Stores block id of the
        // block allocated to a process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

        // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (blockSize[j] >= processSize[i])
                {
                    // allocate block j to p[i] process
                    allocation[i] = j;

                    // Reduce available memory in this block.
                    blockSize[j] -= processSize[i];

                    break;
                }
            }
        }

        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            System.out.print(" " + (i+1) + "\t\t" +
                             processSize[i] + "\t\t");
            if (allocation[i] != -1)
                System.out.print(allocation[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
```

```
    }

    // Driver Code
    public static void main(String[] args)
    {
        int blockSize[] = {100, 500, 200, 300, 600};
        int processSize[] = {212, 417, 112, 426};
        int m = blockSize.length;
        int n = processSize.length;

        firstFit(blockSize, m, processSize, n);
    }
}
```

C#

```
 // C# implementation of First - Fit algorithm
using System;

class GFG
{
    // Method to allocate memory to
    // blocks as per First fit algorithm
    static void firstFit(int []blockSize, int m,
                         int []processSize, int n)
    {
        // Stores block id of the block
        // allocated to a process
        int []allocation = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.Length; i++)
            allocation[i] = -1;

        // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (blockSize[j] >= processSize[i])
                {
                    // allocate block j to p[i] process
                    allocation[i] = j;

                    // Reduce available memory in this block.
                    blockSize[j] -= processSize[i];
```

```
                    break;
                }
            }
        }

    Console.WriteLine("\nProcess No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            Console.Write(" " + (i+1) + "\t\t" +
                          processSize[i] + "\t\t");
            if (allocation[i] != -1)
            Console.Write(allocation[i] + 1);
            else
                Console.Write("Not Allocated");
            Console.WriteLine();
        }
    }

    // Driver Code
    public static void Main()
    {
        int []blockSize = {100, 500, 200, 300, 600};
        int []processSize = {212, 417, 112, 426};
        int m = blockSize.Length;
        int n = processSize.Length;

        firstFit(blockSize, m, processSize, n);
    }
}

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

Output :

```
Process No.    Process Size    Block no.
   1        212         2
   2        417         5
   3        112         2
   4        426         Not Allocated
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/

Chapter 119

Program for Optimal Page Replacement Algorithm

Program for Optimal Page Replacement Algorithm - GeeksforGeeks

Prerequisite: Page Replacement Algorithms

In operating systems, whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.
In this algorithm, OS replaces the page that will not be used for the longest period of time in future.

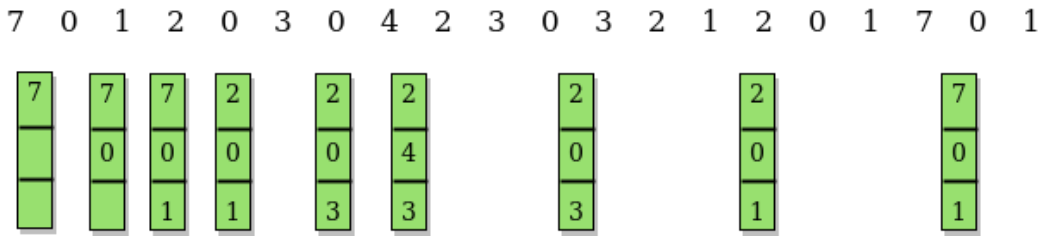
Examples :

```
Input : Number of frames, fn = 3
        Reference String, pg[] = {7, 0, 1, 2,
               0, 3, 0, 4, 2, 3, 0, 3, 2, 1,
               2, 0, 1, 7, 0, 1};
Output : No. of hits = 11
         No. of misses = 9

Input : Number of frames, fn = 4
        Reference String, pg[] = {7, 0, 1, 2,
                  0, 3, 0, 4, 2, 3, 0, 3, 2};
Output : No. of hits = 7
         No. of misses = 6
```

The idea is simple, for every reference we do following :

1. If referred page is already present, increment hit count.

2. If not present, find if a page that is never referenced in future. If such a page exists, replace this page with new page. If no such page exists, find a page that is referenced farthest in future. Replace this page with new page.




```
 // CPP program to demonstrate optimal page
// replacement algorithm.
#include <bits/stdc++.h>
using namespace std;

// Function to check whether a page exists
// in a frame or not
bool search(int key, vector<int>& fr)
{
    for (int i = 0; i < fr.size(); i++)
        if (fr[i] == key)
            return true;
    return false;
}

// Function to find the frame that will not be used
// recently in future after given index in pg[0..pn-1]
int predict(int pg[], vector<int>& fr, int pn, int index)
{
    // Store the index of pages which are going
    // to be used recently in future
    int res = -1, farthest = index;
    for (int i = 0; i < fr.size(); i++) {
        int j;
        for (j = index; j < pn; j++) {
            if (fr[i] == pg[j]) {
                if (j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }

        // If a page is never referenced in future,
```

```
        // return it.
        if (j == pn)
            return i;
    }

    // If all of the frames were not in future,
    // return any of them, we return 0. Otherwise
    // we return res.
    return (res == -1) ? 0 : res;
}

void optimalPage(int pg[], int pn, int fn)
{
    // Create an array for given number of
    // frames and initialize it as empty.
    vector<int> fr;

    // Traverse through page reference array
    // and check for miss and hit.
    int hit = 0;
    for (int i = 0; i < pn; i++) {

        // Page found in a frame : HIT
        if (search(pg[i], fr)) {
            hit++;
            continue;
        }

        // Page not found in a frame : MISS

        // If there is space available in frames.
        if (fr.size() < fn)
            fr.push_back(pg[i]);

        // Find the page to be replaced.
        else {
            int j = predict(pg, fr, pn, i + 1);
            fr[j] = pg[i];
        }
    }
    cout << "No. of hits = " << hit << endl;
    cout << "No. of misses = " << pn - hit << endl;
}

// Driver Function
int main()
{
    int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
```

```
    int pn = sizeof(pg) / sizeof(pg[0]);
    int fn = 4;
    optimalPage(pg, pn, fn);
    return 0;
}
```

Output:

```
No. of hits = 7
No. of misses = 6
```

- The above implementation can optimized using hashing. We can use an unordered_set in place of vector so that search operation can be done in O(1) time.
- Note that optimal page replacement algorithm is not practical as we cannot predict future. However it is used as a reference for other page replacement algorithms.

Source

https://www.geeksforgeeks.org/program-optimal-page-replacement-algorithm/

Chapter 120

Program for Page Replacement Algorithms Set 1 (LRU)

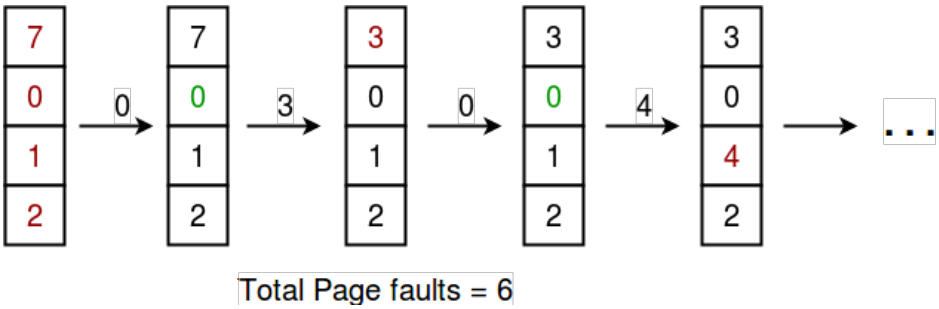
Program for Page Replacement Algorithms Set 1 (LRU) - GeeksforGeeks

Prerequisite: Page Replacement Algorithms

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

In **L**east **R**ecently **U**sed (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely

Let say the page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 . Initially we have 4 page slots empty.
Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.



Given memory capacity (as number of pages it can hold) and a string representing pages to be referred, write a function to find number of page faults.

```
Let capacity be the number of pages that
memory can hold.  Let set be the current
set of pages in memory.

1- Start traversing the pages.
 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size  of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the recent occurred
      index of each page in a map called indexes.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) Find the page in the set that was least
     recently used. We find it using index array.
     We basically need to replace the page with
     minimum index.
     b) Replace the found page with current page.
     c) Increment page faults.
     d) Update index of current page.

2. Return page faults.
```

Below is implementation of above steps.

C++

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

// Function to find page faults using indexes
```

```
int pageFaults(int pages[], int n, int capacity)
{
    // To represent set of current pages. We use
    // an unordered_set so that we quickly check
    // if a page is present in set or not
    unordered_set<int> s;

    // To store least recently used indexes
    // of pages.
    unordered_map<int, int> indexes;

    // Start from initial page
    int page_faults = 0;
    for (int i=0; i<n; i++)
    {
        // Check if the set can hold more pages
        if (s.size() < capacity)
        {
            // Insert it into set if not present
            // already which represents page fault
            if (s.find(pages[i])==s.end())
            {
                s.insert(pages[i]);

                // increment page fault
                page_faults++;
            }

            // Store the recently used index of
            // each page
            indexes[pages[i]] = i;
        }

        // If the set is full then need to perform lru
        // i.e. remove the least recently used page
        // and insert the current page
        else
        {
            // Check if current page is not already
            // present in the set
            if (s.find(pages[i]) == s.end())
            {
                // Find the least recently used pages
                // that is present in the set
                int lru = INT_MAX, val;
                for (auto it=s.begin(); it!=s.end(); it++)
                {
                    if (indexes[*it] < lru)
```

```
                    {
                        lru = indexes[*it];
                        val = *it;
                    }
                }

                // Remove the indexes page
                s.erase(val);

                // insert the current page
                s.insert(pages[i]);

                // Increment page faults
                page_faults++;
            }

            // Update the current page index
            indexes[pages[i]] = i;
        }
    }

    return page_faults;
}

// Driver code
int main()
{
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    cout << pageFaults(pages, n, capacity);
    return 0;
}
```

Java

```
 // Java implementation of above algorithm

import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;

class Test
{
    // Method to find page faults using indexes
    static int pageFaults(int pages[], int n, int capacity)
    {
        // To represent set of current pages. We use
```

```
// an unordered_set so that we quickly check
// if a page is present in set or not
HashSet<Integer> s = new HashSet<>(capacity);

// To store least recently used indexes
// of pages.
HashMap<Integer, Integer> indexes = new HashMap<>();

// Start from initial page
int page_faults = 0;
for (int i=0; i<n; i++)
{
    // Check if the set can hold more pages
    if (s.size() < capacity)
    {
        // Insert it into set if not present
        // already which represents page fault
        if (!s.contains(pages[i]))
        {
            s.add(pages[i]);

            // increment page fault
            page_faults++;
        }

        // Store the recently used index of
        // each page
        indexes.put(pages[i], i);
    }

    // If the set is full then need to perform lru
    // i.e. remove the least recently used page
    // and insert the current page
    else
    {
        // Check if current page is not already
        // present in the set
        if (!s.contains(pages[i]))
        {
            // Find the least recently used pages
            // that is present in the set
            int lru = Integer.MAX_VALUE, val=Integer.MIN_VALUE;

            Iterator<Integer> itr = s.iterator();

            while (itr.hasNext()) {
                int temp = itr.next();
                if (indexes.get(temp) < lru)
```

```
                        {
                            lru = indexes.get(temp);
                            val = temp;
                        }
                    }

                    // Remove the indexes page
                    s.remove(val);

                    // insert the current page
                    s.add(pages[i]);

                    // Increment page faults
                    page_faults++;
                }

                // Update the current page index
                indexes.put(pages[i], i);
            }
        }

        return page_faults;
    }

    // Driver method
    public static void main(String args[])
    {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

        int capacity = 4;

        System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
// This code is contributed by Gaurav Miglani
```

Output:

```
6
```

Note : We can also find the number of page hits. Just have to maintain a separate count. If the current page is already in the memory then that must be count as Page-hit.

We will discuss other Page-replacement Algorithms in further sets.

Source

https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-1-lru/

Chapter 121

Program for Page Replacement Algorithms Set 2 (FIFO)

Program for Page Replacement Algorithms Set 2 (FIFO) - GeeksforGeeks

Prerequisite : Page Replacement Algorithms

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

First In First Out (FIFO) page replacement algorithm –
This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example -1. Consider page reference string 1, 3, 0, 3, 5, 6 and 3 page slots.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> 3 **Page Faults.**
when 3 comes, it is already in memory so —> 0 Page Faults.
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 **Page Fault.**
Finally 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 **Page Fault.**

So total page faults = **5**.

Example -2. Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1.

Using FIFO page replacement algorithm –

0	2	1	6	4	0	1	0	3	1	2	1
0	0	0	0	4	4			4	4	2	
	2	2	2	2	0		hit	0	0	0	
		1	1	1	1	hit		3	3	3	
			6	6	6			6	1	1	hit

So, total number of page faults = 9.

Given memory capacity (as number of pages it can hold) and a string representing pages to be referred, write a function to find number of page faults.

Implementation – Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

```
1- Start traversing the pages.
 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size  of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the pages in the
      queue to perform FIFO.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) Remove the first page from the queue
        as it was the first to be entered in
        the memory
     b) Replace the first page in the queue with
        the current page in the string.
     c) Store current page in the queue.
     d) Increment page faults.

2. Return page faults.
```

C++

```
 // C++ implementation of FIFO page replacement
// in Operating Systems.
#include<bits/stdc++.h>
using namespace std;

// Function to find page faults using FIFO
int pageFaults(int pages[], int n, int capacity)
{
    // To represent set of current pages. We use
    // an unordered_set so that we quickly check
    // if a page is present in set or not
    unordered_set<int> s;
```

```
// To store the pages in FIFO manner
queue<int> indexes;

// Start from initial page
int page_faults = 0;
for (int i=0; i<n; i++)
{
    // Check if the set can hold more pages
    if (s.size() < capacity)
    {
        // Insert it into set if not present
        // already which represents page fault
        if (s.find(pages[i])==s.end())
        {
            s.insert(pages[i]);

            // increment page fault
            page_faults++;

            // Push the current page into the queue
            indexes.push(pages[i]);
        }
    }

    // If the set is full then need to perform FIFO
    // i.e. remove the first page of the queue from
    // set and queue both and insert the current page
    else
    {
        // Check if current page is not already
        // present in the set
        if (s.find(pages[i]) == s.end())
        {
            //Pop the first page from the queue
            int val = indexes.front();

            indexes.pop();

            // Remove the indexes page
            s.erase(val);

            // insert the current page
            s.insert(pages[i]);

            // push the current page into
            // the queue
            indexes.push(pages[i]);
```

```
                // Increment page faults
                page_faults++;
            }
        }
    }

    return page_faults;
}

// Driver code
int main()
{
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,
                   2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    cout << pageFaults(pages, n, capacity);
    return 0;
}
```

Java

```
 // Java implementation of FIFO page replacement
// in Operating Systems.

import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;


class Test
{
    // Method to find page faults using FIFO
    static int pageFaults(int pages[], int n, int capacity)
    {
        // To represent set of current pages. We use
        // an unordered_set so that we quickly check
        // if a page is present in set or not
        HashSet<Integer> s = new HashSet<>(capacity);

        // To store the pages in FIFO manner
        Queue<Integer> indexes = new LinkedList<>() ;

        // Start from initial page
        int page_faults = 0;
        for (int i=0; i<n; i++)
        {
```

```
        // Check if the set can hold more pages
        if (s.size() < capacity)
        {
            // Insert it into set if not present
            // already which represents page fault
            if (!s.contains(pages[i]))
            {
                s.add(pages[i]);

                // increment page fault
                page_faults++;

                // Push the current page into the queue
                indexes.add(pages[i]);
            }
        }

        // If the set is full then need to perform FIFO
        // i.e. remove the first page of the queue from
        // set and queue both and insert the current page
        else
        {
            // Check if current page is not already
            // present in the set
            if (!s.contains(pages[i]))
            {
                //Pop the first page from the queue
                int val = indexes.peek();

                indexes.poll();

                // Remove the indexes page
                s.remove(val);

                // insert the current page
                s.add(pages[i]);

                // push the current page into
                // the queue
                indexes.add(pages[i]);

                // Increment page faults
                page_faults++;
            }
        }
    }

    return page_faults;
```

```
    }

    // Driver method
    public static void main(String args[])
    {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,
                       2, 3, 0, 3, 2};

        int capacity = 4;
        System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
// This code is contributed by Gaurav Miglani
```

Output:

```
7
```

Note – We can also find the number of page hits. Just have to maintain a separate count. If the current page is already in the memory then that must be count as Page-hit.

Belady's anomaly –
Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

Improved By : Villan

Source

https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-2-fifo/

Chapter 122

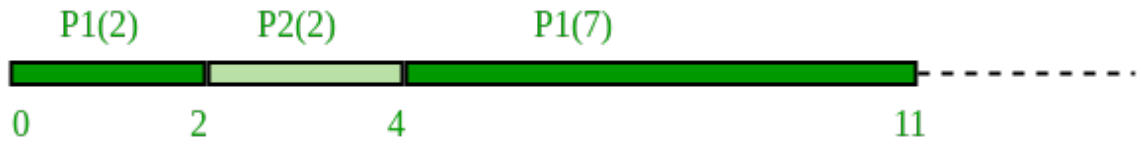
Program for Shortest Job First (SJF) scheduling Set 2 (Preemptive)

Program for Shortest Job First (SJF) scheduling Set 2 (Preemptive) - GeeksforGeeks

In previous post, we have discussed Set 1 of SJF i.e. non-preemptive. In this post we will discuss the preemptive version of SJF known as Shortest Remaining Time First (SRTF).

In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

Process	Duration	Order	Arrival Time
P1	9	1	0
P2	2	2	2



P1 waiting time: 4-2 = 2
P2 waiting time: 0
The average waiting time(AWT): (0 + 2) / 2 = 1

Advantage:
1- Short processes are handled very quickly.

2- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.
3- When a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

Disadvantage:
1- Like shortest job first, it has the potential for process starvation.
2- Long processes may be held off indefinitely if short processes are continually added.

Source:Wiki

```
Implementation:
1- Traverse until all process gets completely
   executed.
   a) Find process with minimum remaining time at
     every single time lap.
   b) Reduce its time by 1.
   c) Check if its remaining time becomes 0
   d) Increment the counter of process completion.
   e) Completion time of current process =
     current_time +1;
   e) Calculate waiting time for each completed
     process.
   wt[i]= Completion time - arrival_time-burst_time
   f)Increment time lap by one.
2- Find turnaround time (waiting_time+burst_time).
```

C/C++

```
 // C++ program to implement Shortest Remaining
// Time First
#include <bits/stdc++.h>
using namespace std;

struct Process {
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time
};

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n,
                                int wt[])
{
    int rt[n];
```

```
// Copy the burst time into rt[]
for (int i = 0; i < n; i++)
    rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT_MAX;
int shortest = 0, finish_time;
bool check = false;

// Process until all processes gets
// completed
while (complete != n) {

    // Find process with minimum
    // remaining time among the
    // processes that arrives till the
    // current time`
    for (int j = 0; j < n; j++) {
        if ((proc[j].art <= t) &&
        (rt[j] < minm) && rt[j] > 0) {
            minm = rt[j];
            shortest = j;
            check = true;
        }
    }

    if (check == false) {
        t++;
        continue;
    }

    // Reduce remaining time by one
    rt[shortest]--;

    // Update minimum
    minm = rt[shortest];
    if (minm == 0)
        minm = INT_MAX;

    // If a process gets completely
    // executed
    if (rt[shortest] == 0) {

        // Increment complete
        complete++;

        // Find finish time of current
        // process
        finish_time = t + 1;
```

```
            // Calculate waiting time
            wt[shortest] = finish_time -
                        proc[shortest].bt -
                        proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        // Increment time
        t++;
    }
}

// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,
                        int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                    total_tat = 0;

    // Function to find waiting time of all
    // processes
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for
    // all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all
    // details
    cout << "Processes "
        << " Burst time "
        << " Waiting time "
        << " Turn around time\n";

    // Calculate total waiting time and
    // total turnaround time
    for (int i = 0; i < n; i++) {
```

```
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
            << proc[i].bt << "\t\t " << wt[i]
            << "\t\t " << tat[i] << endl;
    }

    cout << "\nAverage waiting time = "
        << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },
                    { 3, 7, 2 }, { 4, 3, 3 } };
    int n = sizeof(proc) / sizeof(proc[0]);

    findavgTime(proc, n);
    return 0;
}
```

Java

```
 // Java program to implement Shortest Remaining
// Time First

class Process
{
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time

    public Process(int pid, int bt, int art)
    {
        this.pid = pid;
        this.bt = bt;
        this.art = art;
    }
}

public class GFG
{
    // Method to find the waiting time for all
    // processes
    static void findWaitingTime(Process proc[], int n,
```

```
                                    int wt[])
{
    int rt[] = new int[n];

    // Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = Integer.MAX_VALUE;
    int shortest = 0, finish_time;
    boolean check = false;

    // Process until all processes gets
    // completed
    while (complete != n) {

        // Find process with minimum
        // remaining time among the
        // processes that arrives till the
        // current time`
        for (int j = 0; j < n; j++)
        {
            if ((proc[j].art <= t) &&
              (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false) {
            t++;
            continue;
        }

        // Reduce remaining time by one
        rt[shortest]--;

        // Update minimum
        minm = rt[shortest];
        if (minm == 0)
            minm = Integer.MAX_VALUE;

        // If a process gets completely
        // executed
        if (rt[shortest] == 0) {

            // Increment complete
```

```
            complete++;

            // Find finish time of current
            // process
            finish_time = t + 1;

            // Calculate waiting time
            wt[shortest] = finish_time -
                        proc[shortest].bt -
                        proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        // Increment time
        t++;
    }
}

// Method to calculate turn around time
static void findTurnAroundTime(Process proc[], int n,
                        int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}

// Method to calculate average time
static void findavgTime(Process proc[], int n)
{
    int wt[] = new int[n], tat[] = new int[n];
    int  total_wt = 0, total_tat = 0;

    // Function to find waiting time of all
    // processes
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for
    // all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all
    // details
    System.out.println("Processes " +
                       " Burst time " +
                       " Waiting time " +
```

```
                          " Turn around time");

        // Calculate total waiting time and
        // total turnaround time
        for (int i = 0; i < n; i++) {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            System.out.println(" " + proc[i].pid + "\t\t"
                               + proc[i].bt + "\t\t " + wt[i]
                               + "\t\t" + tat[i]);
        }

        System.out.println("Average waiting time = " +
                           (float)total_wt / (float)n);
        System.out.println("Average turn around time = " +
                           (float)total_tat / (float)n);
    }

    // Driver Method
    public static void main(String[] args)
    {
         Process proc[] = { new Process(1, 6, 1),
                            new Process(2, 8, 1),
                            new Process(3, 7, 2),
                            new Process(4, 3, 3)};

         findavgTime(proc, proc.length);
    }
}
```

Output:

```
Processes  Burst time  Waiting time  Turn around time
 1        6         3        9
 2        8         16        24
 3        7         8        15
 4        3         0        3
Average waiting time = 6.75
Average turn around time = 12.75
```

Source

https://www.geeksforgeeks.org/program-shortest-job-first-scheduling-set-2srtf-make-changesdoneplease-review/

Chapter 123

Program for Shortest Job First (or SJF) scheduling Set 1 (Non-preemptive)

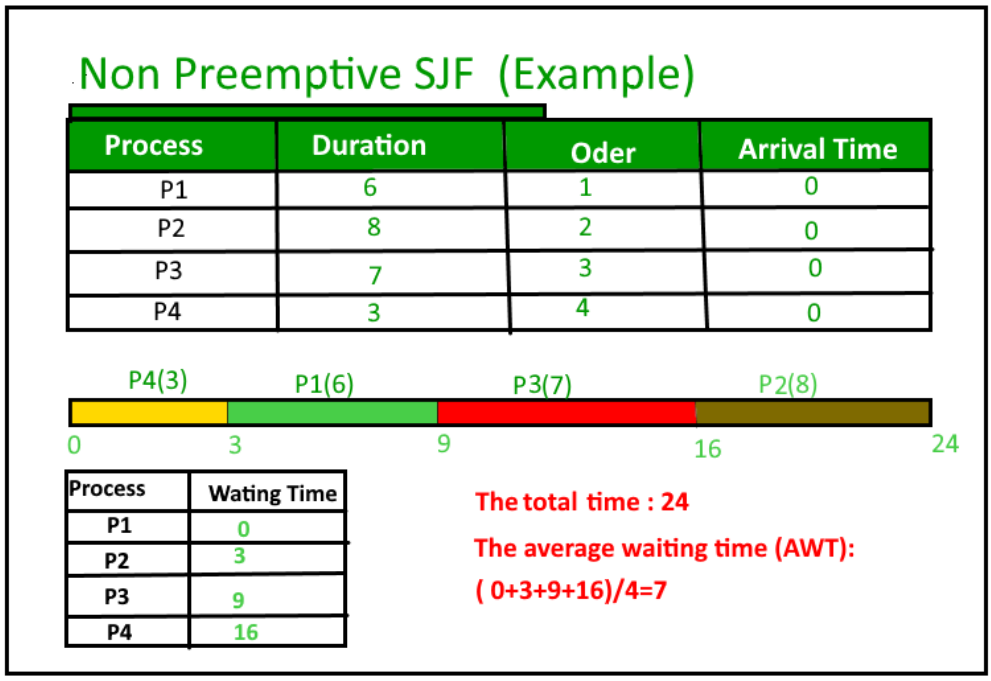
Program for Shortest Job First (or SJF) scheduling Set 1 (Non- preemptive) - GeeksforGeeks

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

- Shortest Job first has the advantage of having minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

Algorithm:

```
1- Sort all the processes in increasing order
   according to burst time.
2- Then simply, apply FCFS.
```



How to compute below times in Round Robin using a program?

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time. Waiting Time = Turn Around Time – Burst Time

In this post, we have assumed arrival times as 0, so turn around and completion times are same.

```
 // C++ program to implement Shortest Job first
#include<bits/stdc++.h>
using namespace std;

struct Process
{
   int pid; // Process ID
   int bt;  // Burst Time
};
```

```
// This function is used for sorting all
// processes in increasing order of burst
// time
bool comparison(Process a, Process b)
{
     return (a.bt < b.bt);
}

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n, int wt[])
{
    // waiting time for first process is 0
    wt[0] = 0;

    // calculating waiting time
    for (int i = 1; i < n ; i++ )
        wt[i] = proc[i-1].bt + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,
                        int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = proc[i].bt + wt[i];
}

//Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all details
    cout << "\nProcesses "<< " Burst time "
         << " Waiting time " << " Turn around time\n";

    // Calculate total waiting time and total turn
    // around time
```

```
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t " << wt[i]
             << "\t\t " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    Process proc[] = {{1, 6}, {2, 8}, {3, 7}, {4, 3}};
    int n = sizeof proc / sizeof proc[0];

    // Sorting processes by burst time.
    sort(proc, proc + n, comparison);

    cout << "Order in which process gets executed\n";
    for (int i = 0 ; i < n; i++)
        cout << proc[i].pid <<" ";

    findavgTime(proc, n);
    return 0;
}
```

Output:

```
Order in which process gets executed
4 1 3 2
Processes  Burst time  Waiting time  Turn around time
 4        3     0         3
 1        6     3         9
 3        7     9         16
 2        8     16         24
Average waiting time = 7
Average turn around time = 13
```

In Set-2 we will discuss the preemptive version of SJF i.e. Shortest Remaining Time First

Source

https://www.geeksforgeeks.org/program-shortest-job-first-sjf-scheduling-set-1-non-preemptive/

Chapter 124

Program for Worst Fit algorithm in Memory Management

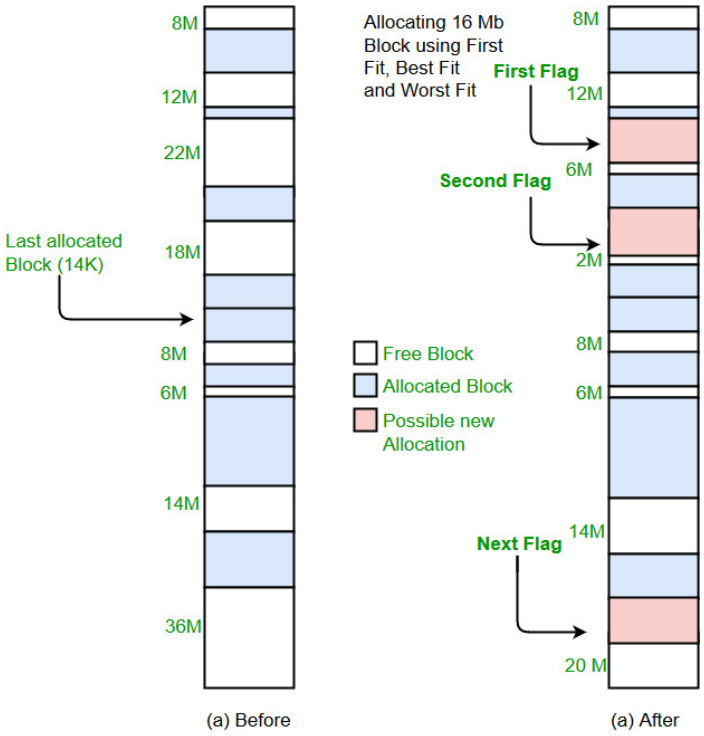
Program for Worst Fit algorithm in Memory Management - GeeksforGeeks

Prerequisite : Partition allocation methods

Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.

Example:

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
   1        212         5
   2        417         2
   3        112         5
   4        426         Not Allocated
```




```
Implementation:
1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the
   maximum block size that can be assigned to
   current process i.e., find max(bockSize[1],
   blockSize[2],.....blockSize[n]) >
   processSize[current], if found then assign
   it to the current process.
5- If not then leave that process and keep checking
   the further processes.
```

Below is implementation of above steps.

C++

```
 // C++ implementation of worst - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to blocks as per worst fit
// algorithm
```

```
void worstFit(int blockSize[], int m, int processSize[],
                                                    int n)
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process
        int wstIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }

        // If we could find a block for current process
        if (wstIdx != -1)
        {
            // allocate block j to p[i] process
            allocation[i] = wstIdx;

            // Reduce available memory in this block.
            blockSize[wstIdx] -= processSize[i];
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
```

```
}

// Driver code
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    worstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

Java

```
 // Java implementation of worst - Fit algorithm

public class GFG
{
    // Method to allocate memory to blocks as per worst fit
    // algorithm
    static void worstFit(int blockSize[], int m, int processSize[],
                                                        int n)
    {
        // Stores block id of the block allocated to a
        // process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

        // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i=0; i<n; i++)
        {
            // Find the best fit block for current process
            int wstIdx = -1;
            for (int j=0; j<m; j++)
            {
                if (blockSize[j] >= processSize[i])
                {
                    if (wstIdx == -1)
                        wstIdx = j;
                    else if (blockSize[wstIdx] < blockSize[j])
                        wstIdx = j;
```

```
                }
            }

            // If we could find a block for current process
            if (wstIdx != -1)
            {
                // allocate block j to p[i] process
                allocation[i] = wstIdx;

                // Reduce available memory in this block.
                blockSize[wstIdx] -= processSize[i];
            }
        }

        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            System.out.print("   " + (i+1) + "\t\t" + processSize[i] + "\t\t");
            if (allocation[i] != -1)
                System.out.print(allocation[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
    }

    // Driver Method
    public static void main(String[] args)
    {
         int blockSize[] = {100, 500, 200, 300, 600};
         int processSize[] = {212, 417, 112, 426};
         int m = blockSize.length;
         int n = processSize.length;

         worstFit(blockSize, m, processSize, n);
    }
}
```

Output:

```
Process No.    Process Size    Block no.
   1        212        5
   2        417        2
   3        112        5
   4        426        Not Allocated
```

Improved By : devansh bhatia 1

Source

https://www.geeksforgeeks.org/program-worst-fit-algorithm-memory-management/

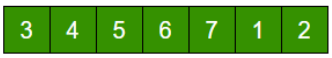
Chapter 125

Program for array rotation

Program for array rotation - GeeksforGeeks

Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.

Rotation of the above array by 2 will make array



METHOD 1 (Using temp array)

```
Input arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2, n =7
1) Store d elements in a temp array
   temp[] = [1, 2]
2) Shift rest of the arr[]
   arr[] = [3, 4, 5, 6, 7, 6, 7]
3) Store back the d elements
   arr[] = [3, 4, 5, 6, 7, 1, 2]
```

Time complexity : O(n)
Auxiliary Space : O(d)

METHOD 2 (Rotate one by one)

```
leftRotate(arr[], d, n)
start
  For i = 0 to i < d
    Left rotate all elements of arr[] by one
end
```

To rotate by one, store arr[0] in a temporary variable temp, move arr[1] to arr[0], arr[2] to arr[1] …and finally temp to arr[n-1]

Let us take the same example arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2
Rotate arr[] by one 2 times
We get [2, 3, 4, 5, 6, 7, 1] after first rotation and [3, 4, 5, 6, 7, 1, 2] after second rotation.

Below is the implementation of the above approach :
C++

```
 // C++ program to rotate an array by
// d elements
#include <bits/stdc++.h>
using namespace std;

/*Function to left Rotate arr[] of
  size n by 1*/
void leftRotatebyOne(int arr[], int n)
{
    int temp = arr[0], i;
    for (i = 0; i < n - 1; i++)
        arr[i] = arr[i + 1];

    arr[i] = temp;
}

/*Function to left rotate arr[] of size n by d*/
void leftRotate(int arr[], int d, int n)
{
    for (int i = 0; i < d; i++)
        leftRotatebyOne(arr, n);
}

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

/* Driver program to test above functions */
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function calling
    leftRotate(arr, 2, n);
    printArray(arr, n);
```

```
    return 0;
}
```

C

```
 // C program to rotate an array by
// d elements
#include <stdio.h>

/* Function to left Rotate arr[] of size n by 1*/
void leftRotatebyOne(int arr[], int n);

/*Function to left rotate arr[] of size n by d*/
void leftRotate(int arr[], int d, int n)
{
    int i;
    for (i = 0; i < d; i++)
        leftRotatebyOne(arr, n);
}

void leftRotatebyOne(int arr[], int n)
{
    int temp = arr[0], i;
    for (i = 0; i < n - 1; i++)
        arr[i] = arr[i + 1];
    arr[i] = temp;
}

/* utility function to print an array */
void printArray(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[] = { 1, 2, 3, 4, 5, 6, 7 };
    leftRotate(arr, 2, 7);
    printArray(arr, 7);
    return 0;
}
```

Java

```
 class RotateArray {
    /*Function to left rotate arr[] of size n by d*/
    void leftRotate(int arr[], int d, int n)
    {
        for (int i = 0; i < d; i++)
            leftRotatebyOne(arr, n);
    }

    void leftRotatebyOne(int arr[], int n)
    {
        int i, temp;
        temp = arr[0];
        for (i = 0; i < n - 1; i++)
            arr[i] = arr[i + 1];
        arr[i] = temp;
    }

    /* utility function to print an array */
    void printArray(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)
    {
        RotateArray rotate = new RotateArray();
        int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
        rotate.leftRotate(arr, 2, 7);
        rotate.printArray(arr, 7);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python3

```
 # Function to left rotate arr[] of size n by d*/
def leftRotate(arr, d, n):
    for i in range(d):
        leftRotatebyOne(arr, n)

# Function to left Rotate arr[] of size n by 1*/
def leftRotatebyOne(arr, n):
    temp = arr[0]
    for i in range(n-1):
        arr[i] = arr[i + 1]
```

```
    arr[n-1] = temp


# utility function to print an array */
def printArray(arr, size):
    for i in range(size):
        print ("% d"% arr[i], end =" ")


# Driver program to test above functions */
arr = [1, 2, 3, 4, 5, 6, 7]
leftRotate(arr, 2, 7)
printArray(arr, 7)

# This code is contributed by Shreyanshi Arun
```

C#

```
 // C# program for array rotation
using System;

class GFG {
    /* Function to left rotate arr[]
    of size n by d*/
    static void leftRotate(int[] arr, int d,
                           int n)
    {
        for (int i = 0; i < d; i++)
            leftRotatebyOne(arr, n);
    }

    static void leftRotatebyOne(int[] arr, int n)
    {
        int i, temp = arr[0];
        for (i = 0; i < n - 1; i++)
            arr[i] = arr[i + 1];

        arr[i] = temp;
    }

    /* utility function to print an array */
    static void printArray(int[] arr, int size)
    {
        for (int i = 0; i < size; i++)
            Console.Write(arr[i] + " ");
    }

    // Driver code
```

```
    public static void Main()
    {
        int[] arr = { 1, 2, 3, 4, 5, 6, 7 };
        leftRotate(arr, 2, 7);
        printArray(arr, 7);
    }
}

// This code is contributed by Sam007
```

Output :

```
3 4 5 6 7 1 2
```

Time complexity : O(n * d)
Auxiliary Space : O(1)

METHOD 3 (A Juggling Algorithm)
This is an extension of method 2. Instead of moving one by one, divide the array in different sets
where number of sets is equal to GCD of n and d and move the elements within sets.
If GCD is 1 as is for the above example array (n = 7 and d =2), then elements will be moved within one set only, we just start with temp = arr[0] and keep moving arr[I+d] to arr[I] and finally store temp at the right place.

Here is an example for n =12 and d = 3. GCD is 3 and

```
Let arr[] be {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

a) Elements are first moved in first set - (See below
   diagram for this movement)



          arr[] after this step --> {4 2 3 7 5 6 10 8 9 1 11 12}

b)    Then in second set.
          arr[] after this step --> {4 5 3 7 8 6 10 11 9 1 2 12}

c)    Finally in third set.
          arr[] after this step --> {4 5 6 7 8 9 10 11 12 1 2 3}
```

Below is the implementation of the above approach :

C++

```
 // C++ program to rotate an array by
```

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

/*Fuction to get gcd of a and b*/
int gcd(int a, int b)
{
    if (b == 0)
        return a;

    else
        return gcd(b, a % b);
}

/*Function to left rotate arr[] of siz n by d*/
void leftRotate(int arr[], int d, int n)
{
    for (int i = 0; i < gcd(d, n); i++) {
        /* move i-th values of blocks */
        int temp = arr[i];
        int j = i;

        while (1) {
            int k = j + d;
            if (k >= n)
                k = k - n;

            if (k == i)
                break;

            arr[j] = arr[k];
            j = k;
        }
        arr[j] = temp;
    }
}

// Function to print an array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
```

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

    // Function calling
    leftRotate(arr, 2, n);
    printArray(arr, n);

    return 0;
}
```

C

```
 #include <stdio.h>

/* function to print an array */
void printArray(int arr[], int size);

/*Fuction to get gcd of a and b*/
int gcd(int a, int b);

/*Function to left rotate arr[] of siz n by d*/
void leftRotate(int arr[], int d, int n)
{
    int i, j, k, temp;
    for (i = 0; i < gcd(d, n); i++) {
        /* move i-th values of blocks */
        temp = arr[i];
        j = i;
        while (1) {
            k = j + d;
            if (k >= n)
                k = k - n;
            if (k == i)
                break;
            arr[j] = arr[k];
            j = k;
        }
        arr[j] = temp;
    }
}

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

```
/*Fuction to get gcd of a and b*/
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
    leftRotate(arr, 2, 7);
    printArray(arr, 7);
    getchar();
    return 0;
}
```

Java

```
 class RotateArray {
    /*Function to left rotate arr[] of siz n by d*/
    void leftRotate(int arr[], int d, int n)
    {
        int i, j, k, temp;
        for (i = 0; i < gcd(d, n); i++) {
            /* move i-th values of blocks */
            temp = arr[i];
            j = i;
            while (true) {
                k = j + d;
                if (k >= n)
                    k = k - n;
                if (k == i)
                    break;
                arr[j] = arr[k];
                j = k;
            }
            arr[j] = temp;
        }
    }

    /*UTILITY FUNCTIONS*/

    /* function to print an array */
    void printArray(int arr[], int size)
```

```
    {
        int i;
        for (i = 0; i < size; i++)
            System.out.print(arr[i] + " ");
    }

    /*Fuction to get gcd of a and b*/
    int gcd(int a, int b)
    {
        if (b == 0)
            return a;
        else
            return gcd(b, a % b);
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        RotateArray rotate = new RotateArray();
        int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
        rotate.leftRotate(arr, 2, 7);
        rotate.printArray(arr, 7);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python3

```
 # Function to left rotate arr[] of size n by d
def leftRotate(arr, d, n):
    for i in range(gcd(d, n)):

        # move i-th values of blocks
        temp = arr[i]
        j = i
        while 1:
            k = j + d
            if k >= n:
                k = k - n
            if k == i:
                break
            arr[j] = arr[k]
            j = k
        arr[j] = temp

# UTILITY FUNCTIONS
# function to print an array
```

```
def printArray(arr, size):
    for i in range(size):
        print ("% d" % arr[i], end =" ")

# Fuction to get gcd of a and b
def gcd(a, b):
    if b == 0:
        return a;
    else:
        return gcd(b, a % b)

# Driver program to test above functions
arr = [1, 2, 3, 4, 5, 6, 7]
leftRotate(arr, 2, 7)
printArray(arr, 7)

# This code is contributed by Shreyanshi Arun
```

C#

```
 // C# program for array rotation
using System;

class GFG {
    /* Function to left rotate arr[]
    of size n by d*/
    static void leftRotate(int[] arr, int d,
                           int n)
    {
        int i, j, k, temp;
        for (i = 0; i < gcd(d, n); i++) {
            /* move i-th values of blocks */
            temp = arr[i];
            j = i;
            while (true) {
                k = j + d;
                if (k >= n)
                    k = k - n;
                if (k == i)
                    break;
                arr[j] = arr[k];
                j = k;
            }
            arr[j] = temp;
        }
    }

    /*UTILITY FUNCTIONS*/
```

```
    /* Function to print an array */
    static void printArray(int[] arr, int size)
    {
        for (int i = 0; i < size; i++)
            Console.Write(arr[i] + " ");
    }

    /* Fuction to get gcd of a and b*/
    static int gcd(int a, int b)
    {
        if (b == 0)
            return a;
        else
            return gcd(b, a % b);
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 1, 2, 3, 4, 5, 6, 7 };
        leftRotate(arr, 2, 7);
        printArray(arr, 7);
    }
}

// This code is contributed by Sam007
```

Output :

```
3 4 5 6 7 1 2
```

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

METHOD 4 (In-Place Rotation)
In this solution, the array is rotated in-place by computing the new index of the current value.

Algorithm:
1. Iterate over array
2. Find the new index of the current value with the formula:
newIndex = (currentIndex – (no_of_rotation % array_length) + array_length) % array_length
3. Swap currentIndex value with newIndex value.
Here is an example for n = 3.
Let arr[] be {0, 1, 2, 3, 4, 5, 6, 7}. The values of array are taken as their index number to make the algorithm more readable and simple to understand.

```
Current array:
Current Index Number -> Value
0                    -> 0
1                    -> 1
2                    -> 2
3                    -> 3
4                    -> 4
5                    -> 5
6                    -> 6
7                    -> 7

Iterate Over the array. Start with 0 index.
Find its new index using the formula:
newIndex = (currentIndex - (no_of_rotation % array_length) + array_length) % array_length
Example:
(0 - (3 % 8) + 8) % 8 = 5

So the new indexes would be:
(0 - (3 % 8) + 8) % 8 = 5
(1 - (3 % 8) + 8) % 8 = 6
(2 - (3 % 8) + 8) % 8 = 7
(3 - (3 % 8) + 8) % 8 = 0
(4 - (3 % 8) + 8) % 8 = 1
(5 - (3 % 8) + 8) % 8 = 2
(6 - (3 % 8) + 8) % 8 = 3
(7 - (3 % 8) + 8) % 8 = 4

New array:
Current Index Number -> New Value
0                    -> 5
1                    -> 6
2                    -> 7
3                    -> 0
4                    -> 1
5                    -> 2
6                    -> 3
7                    -> 4
```

Below is the implementation of the above approach:

```
 // Java Program to rotate an array in-place
import java.util.Arrays;

public class ArrayRotate {

    // main method
    public static void main(String[] args)
```

```
    {
        int[] array = { 1, 2, 3, 4, 5, 6, 7 };
        rotate(array, 2);

        print(array);
    }

    // rotate method
    private static void rotate(int[] array, int n)
    {
        int currIndex = 0, newIndex = 0,
            backupVal = array[currIndex], newVal = array[currIndex];
        int i = 0, arrLen = array.length;
        while (i < arrLen) {
            currIndex = newIndex;

            // compute the new index for current value
            newIndex = (arrLen - (n % arrLen) + currIndex) % arrLen;

            // take backup of new index value
            backupVal = array[newIndex];

            // assign the value to the new index
            array[newIndex] = newVal;

            newVal = backupVal;
            i++;
        }
    }

    // method to print the array
    private static void print(int[] array)
    {
        Arrays.stream(array).forEach(a -> System.out.print(a + " "));
    }
}

// This code is contributed by Shantanoo Sinha
// & siddharth.india51088@gmail.com
```

Output :

```
3 4 5 6 7 1 2
```

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

Please see following posts for other methods of array rotation:
Block swap algorithm for array rotation
Reversal algorithm for array rotation

Improved By : shantanoo, SukhjashanSingh

Source

https://www.geeksforgeeks.org/array-rotation/

Chapter 126

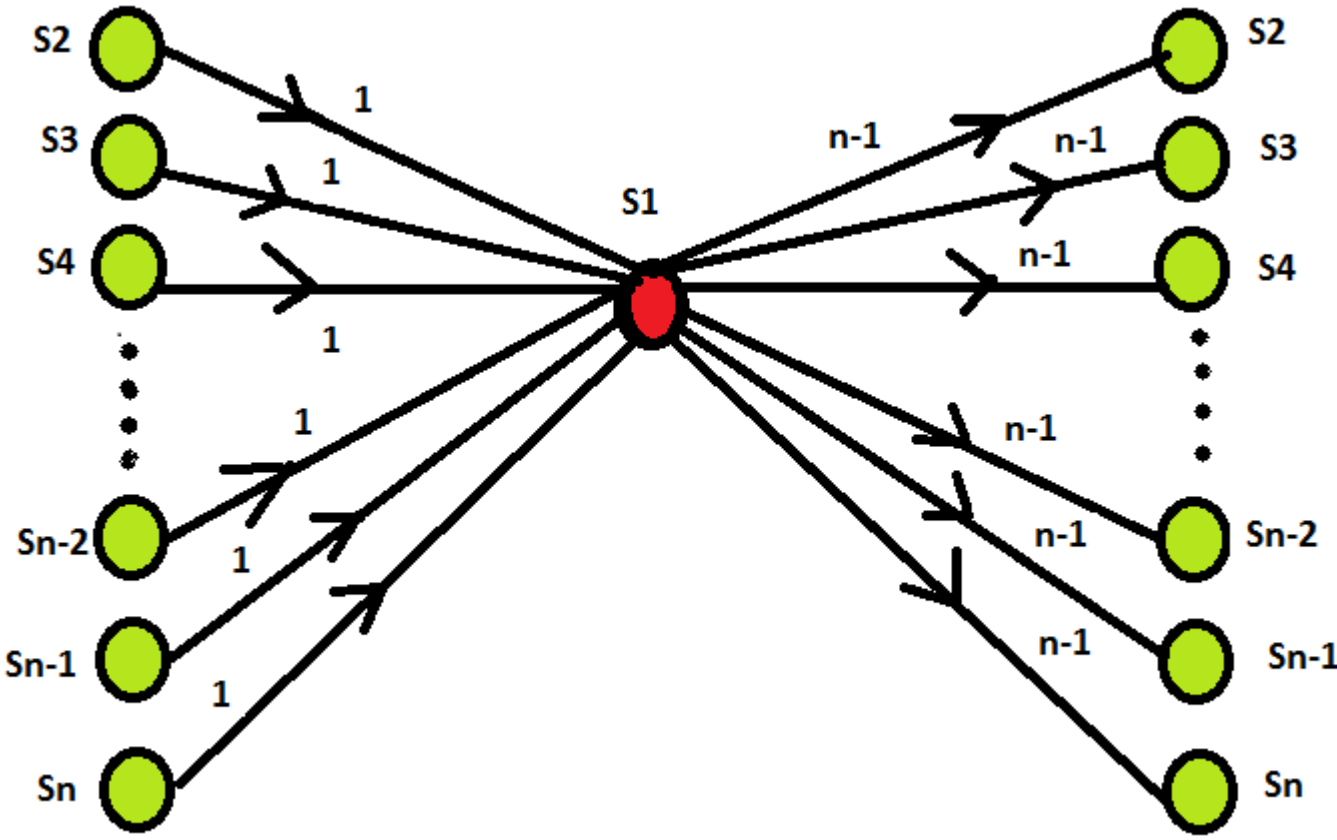
Puzzle Message Spreading

Puzzle Message Spreading - GeeksforGeeks

There are n students in a class, each in possession of a different funny story. As the students were getting bored in the class, they decided to come up with a game so that they can pass their time. They want to share the funny stories with each other by sending electronic messages. Assume that a sender includes all the funny stories he or she knows at the time the message is sent and that a message may only have one addressee. What is the minimum number of messages they need to send to guarantee that everyone of them gets all the funny stories?

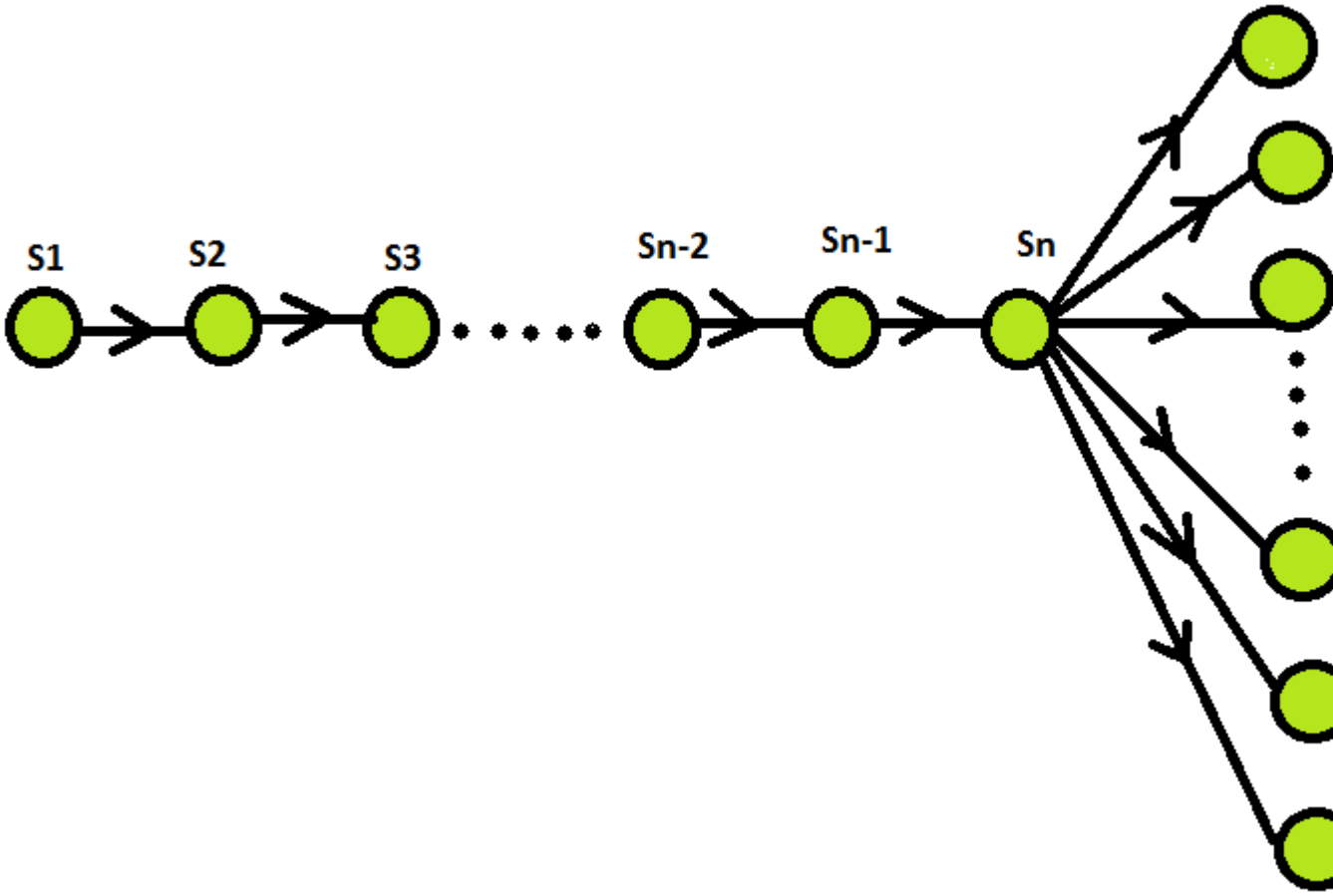
Solution : The minimum number of messages is equal to 2n – 2. There are several ways to do this.

Method 1 : The students can designate one student, say, student 1, to whom everybody else sends the message with the funny story they know. After receiving all these messages, student 1 combines all the funny stories with his or her funny story and send this combined message to each of the other n – 1 students. It can be understood by the below figure. Represent the n students as S1, S2, S3, ……., Sn. Students designate S1 to whom every other student sends the message with the funny story they know.



After receiving all these messages, student 1 combines all the funny stories with his or her funny story and send this combined message to each of the other n – 1 students. Hence, the minimum number of messages is equal to 2n – 2.

Method 2 (Greedy Algorithm): Number the students from 1 to n as S1, S2, S3,……, Sn and send the first n – 1 messages as follows : from S1 to S2, from S2 to S3, and so on, until the message combining the funny stories initially known to students S1, S2, . . . , Sn – 1 is sent to person n. Then send the message combining all the n funny stories from student n i.e, Sn to students S1, S2, . . . , Sn – 1.



Hence, The minimum number of messages is equal to 2n – 2.

Note : The fact that 2n – 2 messages is the smallest number needed to solve the puzzle stems from the fact that an increase of the number of students by one requires at least two extra messages i.e, to and from the extra student, exactly what the above methods provide.

Reference : Algorithmic Puzzles – Anany Levitin, Maria Levitin

Source

https://www.geeksforgeeks.org/puzzle-message-spreading/

Chapter 127

Rearrange a string so that all same characters become d distance away

Rearrange a string so that all same characters become d distance away - GeeksforGeeks

Given a string and a positive integer d. Some characters may be repeated in the given string. Rearrange characters of the given string such that the same characters become d distance away from each other. Note that there can be many possible rearrangements, the output should be one of the possible rearrangements. If no such arrangement is possible, that should also be reported.
Expected time complexity is O(n) where n is length of input string.

```
Examples:
Input:  "abb", d = 2
Output: "bab"

Input:  "aacbbc", d = 3
Output: "abcabc"

Input: "geeksforgeeks", d = 3
Output: egkegkesfesor

Input:  "aaa",  d = 2
Output: Cannot be rearranged
```

Hint: Alphabet size may be assumed as constant (256) and extra space may be used.

Solution: The idea is to count frequencies of all characters and consider the most frequent character first and place all occurrences of it as close as possible. After the most frequent character is placed, repeat the same process for remaining characters.

1) Let the given string be str and size of string be n

2) Traverse str, store all characters and their frequencies in a Max Heap MH. The value of frequency decides the order in MH, i.e., the most frequent character is at the root of MH.

3) Make all characters of str as '\0'.

4) Do following while MH is not empty.
…**a)** Extract the Most frequent character. Let the extracted character be x and its frequency be f.
…**b)** Find the first available position in str, i.e., find the first '\0' in str.
…**c)** Let the first position be p. Fill x at p, p+d,.. p+(f-1)d

Following are C++ and Python implementations of above algorithm.

C/C++

```
 // C++ program to rearrange a string so that all same
// characters become at least d distance away
#include <iostream>
#include <cstring>
#include <cstdlib>
#define MAX 256
using namespace std;

// A structure to store a character 'c' and its frequency 'f'
// in input string
struct charFreq {
    char c;
    int f;
};

// A utility function to swap two charFreq items.
void swap(charFreq *x, charFreq *y) {
    charFreq z = *x;
    *x = *y;
    *y = z;
}

// A utility function to maxheapify the node freq[i] of a heap
// stored in freq[]
void maxHeapify(charFreq freq[], int i, int heap_size)
{
    int l = i*2 + 1;
    int r = i*2 + 2;
    int largest = i;
    if (l < heap_size && freq[l].f > freq[i].f)
        largest = l;
    if (r < heap_size && freq[r].f > freq[largest].f)
        largest = r;
```

```
    if (largest != i)
    {
        swap(&freq[i], &freq[largest]);
        maxHeapify(freq, largest, heap_size);
    }
}

// A utility function to convert the array freq[] to a max heap
void buildHeap(charFreq freq[], int n)
{
    int i = (n - 1)/2;
    while (i >= 0)
    {
        maxHeapify(freq, i, n);
        i--;
    }
}

// A utility function to remove the max item or root from max heap
charFreq extractMax(charFreq freq[], int heap_size)
{
    charFreq root = freq[0];
    if (heap_size > 1)
    {
        freq[0] = freq[heap_size-1];
        maxHeapify(freq, 0, heap_size-1);
    }
    return root;
}

// The main function that rearranges input string 'str' such that
// two same characters become d distance away
void rearrange(char str[], int d)
{
    // Find length of input string
    int n = strlen(str);

    // Create an array to store all characters and their
    // frequencies in str[]
    charFreq freq[MAX] = {{0, 0}};

    int m = 0; // To store count of distinct characters in str[]

    // Traverse the input string and store frequencies of all
    // characters in freq[] array.
    for (int i = 0; i < n; i++)
    {
        char x = str[i];
```

```
        // If this character has occurred first time, increment m
        if (freq[x].c == 0)
            freq[x].c = x, m++;

        (freq[x].f)++;
        str[i] = '\0';  // This change is used later
    }

    // Build a max heap of all characters
    buildHeap(freq, MAX);

    // Now one by one extract all distinct characters from max heap
    // and put them back in str[] with the d distance constraint
    for (int i = 0; i < m; i++)
    {
        charFreq x = extractMax(freq, MAX-i);

        // Find the first available position in str[]
        int p = i;
        while (str[p] != '\0')
            p++;

        // Fill x.c at p, p+d, p+2d, .. p+(f-1)d
        for (int k = 0; k < x.f; k++)
        {
            // If the index goes beyond size, then string cannot
            // be rearranged.
            if (p + d*k >= n)
            {
                cout << "Cannot be rearranged";
                exit(0);
            }
            str[p + d*k] = x.c;
        }
    }
}

// Driver program to test above functions
int main()
{
    char str[] = "aabbcc";
    rearrange(str, 3);
    cout << str;
}
```

Python

```
 // Python program to rearrange a string so that all same
// characters become at least d distance away
MAX = 256

# A structure to store a character 'c' and its frequency 'f'
# in input string
class charFreq(object):
    def __init__(self,c,f):
        self.c = c
        self.f = f

# A utility function to swap two charFreq items.
def swap(x, y):
    return y, x

# A utility function
def toList(string):
    t = []
    for x in string:
        t.append(x)

    return t

# A utility function
def toString(l):
    return ''.join(l)

# A utility function to maxheapify the node freq[i] of a heap
# stored in freq[]
def maxHeapify(freq, i, heap_size):
    l = i*2 + 1
    r = i*2 + 2
    largest = i
    if l < heap_size and freq[l].f > freq[i].f:
        largest = l
    if r < heap_size and freq[r].f > freq[largest].f:
        largest = r
    if largest != i:
        freq[i], freq[largest] = swap(freq[i], freq[largest])
        maxHeapify(freq, largest, heap_size)

# A utility function to convert the array freq[] to a max heap
def buildHeap(freq, n):
    i = (n - 1)/2
    while i >= 0:
        maxHeapify(freq, i, n)
        i-=1
```

```
# A utility function to remove the max item or root from max heap
def extractMax(freq, heap_size):
    root = freq[0]
    if heap_size > 1:
        freq[0] = freq[heap_size-1]
        maxHeapify(freq, 0, heap_size-1)

    return root

# The main function that rearranges input string 'str' such that
# two same characters become d distance away
def rearrange(string, d):
    # Find length of input string
    n = len(string)

    # Create an array to store all characters and their
    # frequencies in str[]
    freq = []
    for x in xrange(MAX):
        freq.append(charFreq(0,0))

    m = 0

    # Traverse the input string and store frequencies of all
    # characters in freq[] array.
    for i in xrange(n):
        x = ord(string[i])

        # If this character has occurred first time, increment m
        if freq[x].c == 0:
            freq[x].c = chr(x)
            m+=1

        freq[x].f+=1
        string[i] = '\0'

    # Build a max heap of all characters
    buildHeap(freq, MAX)

    # Now one by one extract all distinct characters from max heap
    # and put them back in str[] with the d distance constraint
    for i in xrange(m):
        x = extractMax(freq, MAX-i)

        # Find the first available position in str[]
        p = i
        while string[p] != '\0':
            p+=1
```

```
        # Fill x.c at p, p+d, p+2d, .. p+(f-1)d
        for k in xrange(x.f):

            # If the index goes beyond size, then string cannot
            # be rearranged.
            if p + d*k >= n:
                print "Cannot be rearranged"
                return

            string[p + d*k] = x.c

    return toString(string)

# Driver program
string = "aabbcc"
print rearrange(toList(string), 3)

# This code is contributed by BHAVYA JAIN
```

Output:

```
abcabc
```

Algorithmic Paradigm: Greedy Algorithm

Time Complexity: Time complexity of above implementation is O(n + mLog(MAX)). Here n is the length of str, m is count of distinct characters in str[] and MAX is maximum possible different characters. MAX is typically 256 (a constant) and m is smaller than MAX. So the time complexity can be considered as O(n).

More Analysis:
The above code can be optimized to store only m characters in heap, we have kept it this way to keep the code simple. So the time complexity can be improved to O(n + mLogm). It doesn't much matter through as MAX is a constant.

Also, the above algorithm can be implemented using a O(mLogm) sorting algorithm. The first steps of above algorithm remain same. Instead of building a heap, we can sort the freq[] array in non-increasing order of frequencies and then consider all characters one by one from sorted array.

We will soon be covering an extended version where same characters should be moved at least d distance away.

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

Source

https://www.geeksforgeeks.org/rearrange-a-string-so-that-all-same-characters-become-at-least-d-distance-away/

Chapter 128

Rearrange characters in a string such that no two adjacent are same

Rearrange characters in a string such that no two adjacent are same - GeeksforGeeks

Given a string with repeated characters, task is rearrange characters in a string so that no two adjacent characters are same.

Note : It may be assumed that the string has only lowercase English alphabets.

Examples:

```
Input: aaabc
Output: abaca

Input: aaabb
Output: ababa

Input: aa
Output: Not Possible

Input: aaaabc
Output: Not Possible
```

Asked In : Amazon Interview
Prerequisite : priority_queue.

The idea is to put highest frequency character first (a greedy approach). We use a priority queue (Or Binary Max Heap) and put all characters and ordered by their frequencies (highest frequency character at root). We one by one take highest frequency character from the heap

and add it to result. After we add, we decrease frequency of the character and we temporarily move this character out of priority queue so that it is not picked next time.

We have to follow the step to solve this problem, they are:
1. Build a Priority_queue or max_heap, **pq** that stores characters and their frequencies.
..... Priority_queue or max_heap is built on the bases of frequency of character.
2. Create a temporary Key that will used as the previous visited element (previous element in resultant string. Initialize it { char = '#' , freq = '-1' }
3. While **pq** is not empty.
..... Pop an element and add it to result.
..... Decrease frequency of the popped element by '1'
..... Push the previous element back into the priority_queue if it's frequency > '0'
..... Make the current element as previous element for the next iteration.
4. If length of resultant string and original, print "not possible". Else print result.

Below c++ implementation of above idea

```
 // C++ program to rearrange characters in a string
// so that no two adjacent characters are same.
#include<bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

struct Key
{
    int freq; // store frequency of character
    char ch;

    // function for priority_queue to store Key
    // according to freq
    bool operator<(const Key &k) const
    {
        return freq < k.freq;
    }
};

// Function to rearrange character of a string
// so that no char repeat twice
void rearrangeString(string str)
{
    int n = str.length();

    // Store frequencies of all characters in string
    int count[MAX_CHAR] = {0};
    for (int i = 0 ; i < n ; i++)
        count[str[i]-'a']++;

    // Insert all characters with their frequencies
```

```
    // into a priority_queue
    priority_queue< Key > pq;
    for (char c = 'a' ; c <= 'z' ; c++)
        if (count[c-'a'])
            pq.push( Key { count[c-'a'], c} );

    // 'str' that will store resultant value
    str = "" ;

    // work as the previous visited element
    // initial previous element be. ( '#' and
    // it's frequency '-1' )
    Key prev {-1, '#'} ;

    // traverse queue
    while (!pq.empty())
    {
        // pop top element from queue and add it
        // to string.
        Key k = pq.top();
        pq.pop();
        str = str + k.ch;

        // IF frequency of previous character is less
        // than zero that means it is useless, we
        // need not to push it
        if (prev.freq > 0)
            pq.push(prev);

        // make current character as the previous 'char'
        // decrease frequency by 'one'
        (k.freq)--;
        prev = k;
    }

    // If length of the resultant string and original
    // string is not same then string is not valid
    if (n != str.length())
        cout << " Not valid String " << endl;

    else // valid string
        cout << str << endl;
}

// Driver program to test above function
int main()
{
    string str = "bbbaa" ;
```

```
    rearrangeString(str);
    return 0;
}
```

Output:

```
babab
```

Time complexity : O(n log n)

Source

https://www.geeksforgeeks.org/rearrange-characters-string-no-two-adjacent/

Chapter 129

Reverse Delete Algorithm for Minimum Spanning Tree

Reverse Delete Algorithm for Minimum Spanning Tree - GeeksforGeeks

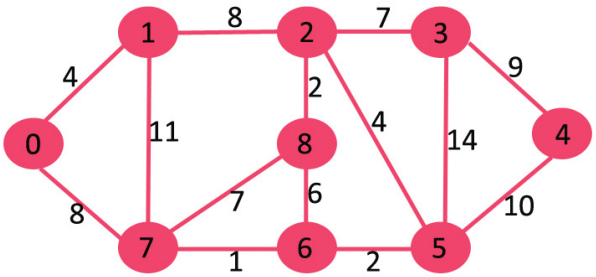
Reverse Delete algorithm is closely related to Kruskal's algorithm. In Kruskal's algorithm what we do is : Sort edges by increasing order of their weights. After sorting, we one by one pick edges in increasing order. We include current picked edge if by including this in spanning tree not form any cycle until there are V-1 edges in spanning tree, where V = number of vertices.

In Reverse Delete algorithm, we sort all edges in **decreasing** order of their weights. After sorting, we one by one pick edges in decreasing order. We **include current picked edge if excluding current edge causes disconnection in current graph**. The main idea is delete edge if its deletion does not lead to disconnection of graph.

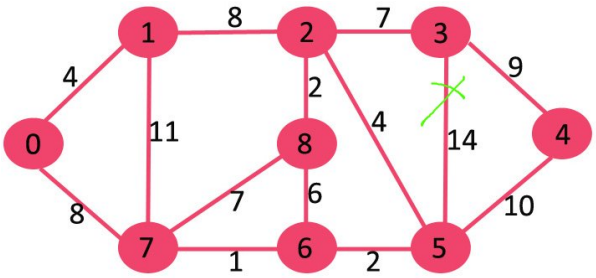
The Algorithm

```
1) Sort all edges of graph in non-increasing order of
   edge weights.

2) Initialize MST as original graph and remove extra
   edges using step 3.

3) Pick highest weight edge from remaining edges and
   check if deleting the edge disconnects the graph
   or not.
       If disconnects, then we don't delete the edge.
       Else we delete the edge and continue.
```

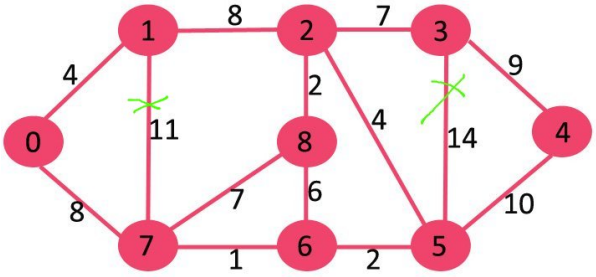
Illustration:
Let us understand with the following example:



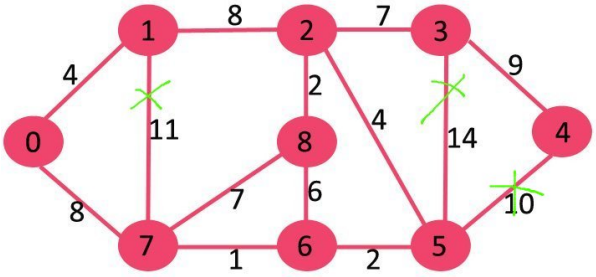
If we delete highest weight edge of weight 14, graph doesn't become disconnected, so we remove it.



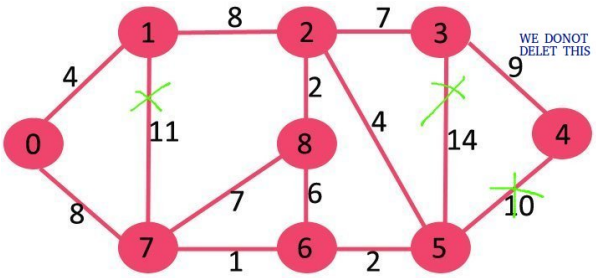
Next we delete 11 as deleting it doesn't disconnect the graph.



Next we delete 10 as deleting it doesn't disconnect the graph.



Next is 9. We cannot delete 9 as deleting it causes disconnection.



We continue this way and following edges remain in final MST.

```
Edges in MST
(3, 4)
(0, 7)
(2, 3)
(2, 5)
(0, 1)
(5, 6)
(2, 8)
(6, 7)
```

Note : In case of same weight edges, we can pick any edge of the same weight edges.

Below is C++ implementation of above steps.

```
 // C++ program to find Minimum Spanning Tree
// of a graph using Reverse Delete Algorithm
#include<bits/stdc++.h>
using namespace std;

// Creating shortcut for an integer pair
typedef  pair<int, int> iPair;

// Graph class represents a directed graph
// using adjacency list representation
class Graph
{
    int V;    // No. of vertices
    list<int> *adj;
    vector< pair<int, iPair> > edges;
    void DFS(int v, bool visited[]);

public:
    Graph(int V);   // Constructor

    // function to add an edge to graph
```

```
    void addEdge(int u, int v, int w);

    // Returns true if graph is connected
    bool isConnected();

    void reverseDeleteMST();
};

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

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

void Graph::DFS(int v, bool visited[])
{
    // Mark the current node as visited and print it
    visited[v] = true;

    // Recur for all the vertices adjacent to
    // this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFS(*i, visited);
}

// Returns true if given graph is connected, else false
bool Graph::isConnected()
{
    bool visited[V];
    memset(visited, false, sizeof(visited));

    // Find all reachable vertices from first vertex
    DFS(0, visited);

    // If set of reachable vertices includes all,
    // return true.
    for (int i=1; i<V; i++)
        if (visited[i] == false)
            return false;
```

```
    return true;
}

// This function assumes that edge (u, v)
// exists in graph or not,
void Graph::reverseDeleteMST()
{
    // Sort edges in increasing order on basis of cost
    sort(edges.begin(), edges.end());

    int mst_wt = 0;  // Initialize weight of MST

    cout << "Edges in MST\n";

    // Iterate through all sorted edges in
    // decreasing order of weights
    for (int i=edges.size()-1; i>=0; i--)
    {
        int u = edges[i].second.first;
        int v = edges[i].second.second;

        // Remove edge from undirected graph
        adj[u].remove(v);
        adj[v].remove(u);

        // Adding the edge back if removing it
        // causes disconnection. In this case this
        // edge becomes part of MST.
        if (isConnected() == false)
        {
            adj[u].push_back(v);
            adj[v].push_back(u);

            // This edge is part of MST
            cout << "(" << u << ", " << v << ") \n";
            mst_wt += edges[i].first;
        }
    }

    cout << "Total weight of MST is " << mst_wt;
}

// Driver code
int main()
{
    // create the graph given in above fugure
    int V = 9;
```

```
    Graph g(V);

    //  making above shown graph
    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    g.reverseDeleteMST();
    return 0;
}
```

Output :

```
Edges in MST
(3, 4)
(0, 7)
(2, 3)
(2, 5)
(0, 1)
(5, 6)
(2, 8)
(6, 7)
Total weight of MST is 37
```

Notes :

1. The above implementation is a simple/naive implementation of Reverse Delete algorithm and can be optimized to O(E log V (log log V)^3) [Source : Wiki]. But this optimized time complexity is still less than Prim and Kruskal Algorithms for MST.
2. The above implementation modifies the original graph. We can create a copy of the graph if original graph must be retained.

References:
https://en.wikipedia.org/wiki/Reverse-delete_algorithm

Source

https://www.geeksforgeeks.org/reverse-delete-algorithm-minimum-spanning-tree/

Chapter 130

Schedule jobs so that each server gets equal load

Schedule jobs so that each server gets equal load - GeeksforGeeks

There are n servers. Each server i is currently processing a(i) amount of requests. There is another array b in which b(i) represents number of incoming requests that are scheduled to server i. Reschedule the incoming requests in such a way that each server i holds equal amount of requests after rescheduling. An incoming request to server i can be rescheduled only to server i-1, i, i+1. If there is no such rescheduling possible then output -1 else print number of requests hold by each server after resheduling.

Examples:

```
Input : a = {6, 14, 21, 1}
        b = {15, 7, 10, 10}
Output : 21
b(0) scheduled to a(0) --> a(0) = 21
b(1) scheduled to a(1) --> a(1) = 21
b(2) scheduled to a(3) --> a(3) = 11
b(3) scheduled to a(3) --> a(3) = 21
a(2) remains unchanged --> a(2) = 21

Input : a = {1, 2, 3}
        b = {1, 100, 3}
Output : -1
No rescheduling will result in equal requests.
```

Approach: Observe that each element of array b is always added to any one element of array a exactly once. Thus sum of all elements of array b + sum of all elements of old array a = sum of all elements of new array a. Let this sum be S. Also all the elements of new array a are equal. Let each new element is x. If array a has n elements, this gives

```
x * n = S
 => x = S/n      ....(1)
```

Thus all the equal elements of new array a is given by eqn(1). Now to make each a(i) equals to x we need to add x-a(i) to each element. We will iterate over entire array a and check whether a(i) can be made equal to x. There are multiple possibilities:
1. a(i) > x: In this case a(i) can never be made equal to x. So output -1.
2. a(i) + b(i) + b(i+1) = x. Simply add b(i) + b(i+1) to a(i) and update b(i), b(i+1) to zero.
3. a(i) + b(i) = x. Add b(i) to a(i) and update b(i) to zero.
4. a(i) + b(i+1) = x. Add b(i+1) to a(i) and update b(i+1) to zero.

After array a is completely traversed, check whether all elements of array b are zero or not. If yes then print a(0) otherwise print -1.

Why b(i) is updated to zero after addition?
Consider a test case in which b(i) is neither added to a(i-1) nor a(i). In that case, we are bounded to add b(i) to a(i+1). Thus while iterating over the array a when we begin performing computations on element a(i), first we add element b(i-1) to a(i) to take into consideration above possibility. Now if b(i-1) is already added to a(i-1) or a(i-2) then in that case it cannot be added to a(i). So to avoid this double addition of b(i) it is updated to zero.

The stepwise algorithm is:

```
1. Compute sum S and find x = S / n
2. Iterate over array a
3. for each element a(i) do:
   a(i) += b(i-1)
   b(i-1) = 0;
   if a(i) > x:
      break
   else:
     check for other three possibilities
     and update a(i) and b(i).
4. Check whether all elements of b(i) are
   zero or not.
```

Implementation:

```
 // CPP program to schedule jobs so that
// each server gets equal load.
#include <bits/stdc++.h>
using namespace std;

// Function to find new array a
int solve(int a[], int b[], int n)
{
```

```
int i;
long long int s = 0;

// find sum S of both arrays a and b.
for (i = 0; i < n; i++)
    s += (a[i] + b[i]);

// Single element case.
if (n == 1)
    return a[0] + b[0];

// This checks whether sum s can be divided
// equally between all array elements. i.e.
// whether all elements can take equal value
// or not.
if (s % n != 0)
    return -1;

// Compute possible value of new array
// elements.
int x = s / n;

for (i = 0; i < n; i++) {

    // Possibility 1
    if (a[i] > x)
        return -1;

    // ensuring that all elements of
    // array b are used.
    if (i > 0) {
        a[i] += b[i - 1];
        b[i - 1] = 0;
    }

    // If a(i) already updated to x
    // move to next element in array a.
    if (a[i] == x)
        continue;

    // Possibility 2
    int y = a[i] + b[i];
    if (i + 1 < n)
        y += b[i + 1];
    if (y == x) {
        a[i] = y;
        b[i] = b[i + 1] = 0;
        continue;
```

```
        }

        // Possibility 3
        if (a[i] + b[i] == x) {
            a[i] += b[i];
            b[i] = 0;
            continue;
        }

        // Possibility 4
        if (i + 1 < n &&
            a[i] + b[i + 1] == x) {
            a[i] += b[i + 1];
            b[i + 1] = 0;
            continue;
        }

        // If a(i) can not be made equal
        // to x even after adding all
        // possible elements from b(i)
        // then print -1.
        return -1;
    }

    // check whether all elements of b
    // are used.
    for (i = 0; i < n; i++)
        if (b[i] != 0)
            return -1;

    // Return the new array element value.
    return x;
}

int main()
{
    int a[] = { 6, 14, 21, 1 };
    int b[] = { 15, 7, 10, 10 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << solve(a, b, n);
    return 0;
}
```

```
  Output: 21
```

Time Complexity: O(n)
Auxiliary Space : O(1) If we are not allowed to modify original arrays, then O(n)

Source

https://www.geeksforgeeks.org/schedule-jobs-server-gets-equal-load/

Chapter 131

Scheduling priority tasks in limited time and minimizing loss

Scheduling priority tasks in limited time and minimizing loss - GeeksforGeeks

Given n tasks with arrival time, priority and number of time units they need. We need to schedule these tasks on a single resource. The objective is to arrange tasks such that maximum priority tasks are taken. Objective is to minimize sum of product of priority and left time of tasks that are not scheduled due to limited time. This criteria simply means that the scheduling should cause minimum possible loss.

Examples:

```
Input : Total time = 3
        Task1: arrival = 1, units = 2, priority = 300
        Task2: arrival = 2, units = 2, priority = 100
Output : 100
Explanation : Two tasks are given and time to finish
them is 3 units. First task arrives at time 1 and it
needs 2 units. Since no other task is running at that
time, we take it. 1 unit of task 1 is over. Now task 2
arrives. The priority of task 1 is more than task 2
(300 > 100) thus 1 more unit  of task 1 is employed.
Now 1 unit of time is left and we have 2 units of task
2 to be done. So simply 1 unit of task 2 is done and
the answer is ( units of task 2 left X priority of
task 2 ) = 1 X 100 = 100

Input : Total Time = 3
        Task1: arrival = 1, units = 1, priority = 100
```

```
        Task2: arrival = 2, units = 2, priority = 300
Output : 0
```

We use a priority queue and schedule one unit of task at a time.

1. Initialize total loss as sum of each priority and units. The idea is to initialize result as maximum, then one by one subtract priorities of tasks that are scheduled.
2. Sort all tasks according to arrival time.
3. Process through each unit of time from 1 to total time. For current time, pick the highest priority task that is available. Schedule 1 unit of this task and subtract its priority from total loss.

Below is C++ implementation of above steps.

```
 // C++ code for task scheduling on basis
// of priority and arrival time
#include "bits/stdc++.h"
using namespace std;

// t is total time. n is number of tasks.
// arriv[] stores arrival times. units[]
// stores units of time required. prior[]
// stores priorities.
int minLoss(int n, int t, int arriv[],
            int units[],  int prior[])
{
    // Calculating maximum possible answer
    // that could be calculated. Later we
    // will subtract the tasks from it
    // accordingly.
    int ans = 0;
    for (int i = 0; i < n; i++)
        ans += prior[i] * units[i];

    // Pushing tasks in a vector so that they
    // could be sorted according with their
    // arrival time
    vector<pair<int, int> > v;
    for (int i = 0; i < n; i++)
        v.push_back({ arriv[i], i });

    // Sorting tasks in vector identified by
    // index and arrival time with respect
    // to their arrival time
    sort(v.begin(), v.end());
```

```
    // Priority queue to hold tasks to be
    // scheduled.
    priority_queue<pair<int, int> > pq;

    // Consider one unit of time at a time.
    int ptr = 0; // index in sorted vector
    for (int i = 1; i <= t; i++) {

        // Push all tasks that have arrived at
        // this time. Note that the tasks that
        // arrived earlier and could not be scheduled
        // are already in pq.
        while (ptr < n and v[ptr].x == i) {
            pq.push({ prior[v[ptr].y], units[v[ptr].y] });
            ptr++;
        }

        // Remove top priority task to be scheduled
        // at time i.
        if (!pq.empty()) {
            auto tmp = pq.top();
            pq.pop();

            // Removing 1 unit of task
            // from answer as we could
            // schedule it.
            ans -= tmp.x;
            tmp.y--;
            if (tmp.y)
                pq.push(tmp);
        }
    }

    return ans;
}

// driver code
int main()
{
    int n = 2, t = 3;
    int arriv[] = { 1, 2 };
    int units[] = { 2, 2 };
    int prior[] = { 100, 300 };

    printf("%d\n",  minLoss(n, t, arriv, units, prior));
    return 0;
}
```

Output:

```
100
```

Source

https://www.geeksforgeeks.org/scheduling-priority-tasks-limited-time-minimizing-loss/

Chapter 132

Set Cover Problem Set 1 (Greedy Approximate Algorithm)

Set Cover Problem Set 1 (Greedy Approximate Algorithm) - GeeksforGeeks

Given a universe U of n elements, a collection of subsets of U say S = {S_1, S_2…,S_m} where every subset S_i has an associated cost. Find a minimum cost subcollection of S that covers all elements of U.

Example:

```
   U = {1,2,3,4,5}
   S = {S1,S2,S3}

   S1 = {4,1,3},    Cost(S1) = 5
   S2 = {2,5},      Cost(S2) = 10
   S3 = {1,4,3,2},  Cost(S3) = 3

Output: Minimum cost of set cover is 13 and
        set cover is {S2, S3}

There are two possible set covers {S1, S2} with cost 15
and {S2, S3} with cost 13.
```

Why is it useful?
It was one of Karp's NP-complete problems, shown to be so in 1972. Other applications: edge covering, vertex cover
Interesting example: IBM finds computer viruses (wikipedia)
Elements- 5000 known viruses

Sets- 9000 substrings of 20 or more consecutive bytes from viruses, not found in 'good' code.
A set cover of 180 was found. It suffices to search for these 180 substrings to verify the existence of known computer viruses.

Another example: Consider General Motors needs to buy a certain amount of varied supplies and there are suppliers that offer various deals for different combinations of materials (Supplier A: 2 tons of steel + 500 tiles for $x; Supplier B: 1 ton of steel + 2000 tiles for $y; etc.). You could use set covering to find the best way to get all the materials while minimizing cost
Source: http://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf

Set Cover is NP-Hard:
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 Logn approximate algorithm.

2-Approximate Greedy Algorithm:
Let U be the universe of elements, $\{S_1, S_2, \ldots S_m\}$ be collection of subsets of U and $Cost(S_1)$, $C(S_2)$, … $Cost(S_m)$ be costs of subsets.

```
1) Let I represents set of elements included so far.  Initialize I = {}

2) Do following while I is not same as U.
    a) Find the set Si in {S1, S2, ... Sm} whose cost effectiveness is
       smallest, i.e., the ratio of cost C(Si) and number of newly added
       elements is minimum.
       Basically we pick the set for which following value is minimum.
           Cost(Si) / |Si - I|
    b) Add elements of above picked Si to I, i.e.,  I = I U Si
```

Example:
Let us consider the above example to understand Greedy Algorithm.

First Iteration:
I = {}

The per new element cost for S_1 = $Cost(S_1)/S_1 - I = 5/3$

The per new element cost for S_2 = $Cost(S_2)/S_2 - I = 10/2$

The per new element cost for S_3 = $Cost(S_3)/S_3 - I = 3/4$

Since S_3 has minimum value S_3 is added, I becomes {1,4,3,2}.

Second Iteration:
I = {1,4,3,2}

The per new element cost for S_1 = $Cost(S_1)/S_1 - I = 5/0$
Note that S_1 doesn't add any new element to I.

The per new element cost for S_2 = $Cost(S_2)/S_2 - I = 10/1$
Note that S_2 adds only 5 to I.

The greedy algorithm provides the optimal solution for above example, but it may not provide optimal solution all the time. Consider the following example.

```
S1 = {1, 2}
S2 = {2, 3, 4, 5}
S3 = {6, 7, 8, 9, 10, 11, 12, 13}
S4 = {1, 3, 5, 7, 9, 11, 13}
S5 = {2, 4, 6, 8, 10, 12, 13}

Let the cost of every set be same.

The greedy algorithm produces result as {S3, S2, S1}

The optimal solution is {S4, S5}
```

Proof that the above greedy algorithm is Logn approximate.
Let OPT be the cost of optimal solution. Say (k-1) elements are covered before an iteration of above greedy algorithm. The cost of the k'th element <= OPT / (n-k+1) (Note that cost of an element is evaluated by cost of its set divided by number of elements added by its set). How did we get this result? Since k'th element is not covered yet, there is a S_i that has not been covered before the current step of greedy algorithm and it is there in OPT. Since greedy algorithm picks the most cost effective S_i, per-element-cost in the picked set must be smaller than OPT divided by remaining elements. Therefore cost of k'th element <= OPT/U-I (Note that U-I is set of not yet covered elements in Greedy Algorithm). The value of U-I is n - (k-1) which is n-k+1.

```
Cost of Greedy Algorithm = Sum of costs of n elements
                         [putting k = 1, 2..n in above formula]
                          <= (OPT/n + OPT(n-1) + ... + OPT/n)
                          <= OPT(1 + 1/2 + ..... 1/n)
                         [Since 1 + 1/2 + .. 1/n   Log n]
                          <= OPT * Logn
```

Source:
http://math.mit.edu/~goemans/18434S06/setcover-tamara.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.

Source

https://www.geeksforgeeks.org/set-cover-problem-set-1-greedy-approximate-algorithm/

Chapter 133

Shortest Superstring Problem

Shortest Superstring Problem - GeeksforGeeks

Given a set of n strings arr[], find the smallest string that contains each string in the given set as substring. We may assume that no string in arr[] is substring of another string.

Examples:

```
Input:  arr[] = {"geeks", "quiz", "for"}
Output: geeksquizfor

Input:  arr[] = {"catg", "ctaagt", "gcta", "ttca", "atgcatc"}
Output: gctaagttcatgcatc
```

Shortest Superstring Greedy Approximate Algorithm
Shortest Superstring Problem is a NP Hard problem. A solution that always finds shortest superstring takes exponential time. Below is an Approximate Greedy algorithm.

```
Let arr[] be given set of strings.

1) Create an auxiliary array of strings, temp[].  Copy contents
   of arr[] to temp[]

2) While temp[] contains more than one strings
     a) Find the most overlapping string pair in temp[]. Let this
        pair be 'a' and 'b'.
     b) Replace 'a' and 'b' with the string obtained after combining
        them.

3) The only string left in temp[] is the result, return it.
```

Two strings are overlapping if prefix of one string is same suffix of other string or vice verse. The maximum overlap mean length of the matching prefix and suffix is maximum.

Working of above Algorithm:

```
arr[] = {"catgc", "ctaagt", "gcta", "ttca", "atgcatc"}
Initialize:
temp[] = {"catgc", "ctaagt", "gcta", "ttca", "atgcatc"}

The most overlapping strings are "catgc" and "atgcatc"
(Suffix of length 4 of "catgc" is same as prefix of "atgcatc")
Replace two strings with "catgcatc", we get
temp[] = {"catgcatc", "ctaagt", "gcta", "ttca"}

The most overlapping strings are "ctaagt" and "gcta"
(Prefix of length 3 of "ctaagt" is same as suffix of "gcta")
Replace two strings with "gctaagt", we get
temp[] = {"catgcatc", "gctaagt", "ttca"}

The most overlapping strings are "catgcatc" and "ttca"
(Prefix of length 2 of "catgcatc" as suffix of "ttca")
Replace two strings with "ttcatgcatc", we get
temp[] = {"ttcatgcatc", "gctaagt"}

Now there are only two strings in temp[], after combing
the two in optimal way, we get tem[] = {"gctaagttcatgcatc"}

Since temp[] has only one string now, return it.
```

Below is C++ implementation of above algorithm.

```
 // C++ program to find shortest superstring using Greedy
// Approximate Algorithm
#include <bits/stdc++.h>
using namespace std;

// Utility function to calculate minimum of two numbers
int min(int a, int b)
{
    return (a < b) ? a : b;
}

// Function to calculate maximum overlap in two given strings
int findOverlappingPair(string str1, string str2, string &str)
{
    // max will store maximum overlap i.e maximum
    // length of the matching prefix and suffix
    int max = INT_MIN;
    int len1 = str1.length();
    int len2 = str2.length();

    // check suffix of str1 matches with prefix of str2
    for (int i = 1; i <= min(len1, len2); i++)
```

```
    {
        // compare last i characters in str1 with first i
        // characters in str2
        if (str1.compare(len1-i, i, str2, 0, i) == 0)
        {
            if (max < i)
            {
                //update max and str
                max = i;
                str = str1 + str2.substr(i);
            }
        }
    }

    // check prefix of str1 matches with suffix of str2
    for (int i = 1; i <= min(len1, len2); i++)
    {
        // compare first i characters in str1 with last i
        // characters in str2
        if (str1.compare(0, i, str2, len2-i, i) == 0)
        {
            if (max < i)
            {
                //update max and str
                max = i;
                str = str2 + str1.substr(i);
            }
        }
    }

    return max;
}

// Function to calculate smallest string that contains
// each string in the given set as substring.
string findShortestSuperstring(string arr[], int len)
{
    // run len-1 times to consider every pair
    while(len != 1)
    {
        int max = INT_MIN;  // to store  maximum overlap
        int l, r;   // to store array index of strings
        // involved in maximum overlap
        string resStr;  // to store resultant string after
        // maximum overlap

        for (int i = 0; i < len; i++)
        {
```

```
            for (int j = i + 1; j < len; j++)
            {
                string str;

                // res will store maximum length of the matching
                // prefix and suffix str is passed by reference and
                // will store the resultant string after maximum
                // overlap of arr[i] and arr[j], if any.
                int res = findOverlappingPair(arr[i], arr[j], str);

                // check for maximum overlap
                if (max < res)
                {
                    max = res;
                    resStr.assign(str);
                    l = i, r = j;
                }
            }
        }

        len--;  //ignore last element in next cycle

        // if no overlap, append arr[len] to arr[0]
        if (max == INT_MIN)
            arr[0] += arr[len];
        else
        {
            arr[l] = resStr;   // copy resultant string to index l
            arr[r] = arr[len];  // copy string at last index to index r
        }
    }
    return arr[0];
}

// Driver program
int main()
{
    string arr[] = {"catgc", "ctaagt", "gcta", "ttca", "atgcatc"};
    int len = sizeof(arr)/sizeof(arr[0]);

    cout << "The Shortest Superstring is "
         << findShortestSuperstring(arr, len);

    return 0;
}
// This code is contributed by Aditya Goel
```

Performance of above algorithm:

The above Greedy Algorithm is proved to be 4 approximate (i.e., length of the superstring generated by this algorithm is never beyond 4 times the shortest possible superstring). This algorithm is conjectured to 2 approximate (nobody has found case where it generates more than twice the worst). Conjectured worst case example is $\{ab^k, b^kc, b^{k+1}\}$. For example {"abb", "bbc", "bbb"}, the above algorithm may generate "abbcbbb" (if "abb" and "bbc" are picked as first pair), but the actual shortest superstring is "abbbc". Here ratio is 7/5, but for large k, ration approaches 2.

There exist better approximate algorithms for this problem. Please refer below link.
Shortest Superstring Problem Set 2 (Using Set Cover)

Applications:
Useful in the genome project since it will allow researchers to determine entire coding regions from a collection of fragmented sections.

Reference:
http://fileadmin.cs.lth.se/cs/Personal/Andrzej_Lingas/superstring.pdf
http://math.mit.edu/~goemans/18434S06/superstring-lele.pdf

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

Source

https://www.geeksforgeeks.org/shortest-superstring-problem/

Chapter 134

Smallest number with sum of digits as N and divisible by 10^N

Smallest number with sum of digits as N and divisible by 10^N - GeeksforGeeks

Find the smallest number such that the sum of its digits is N and it is divisible by 10^N.

Examples :

```
Input : N = 5
Output : 500000
500000 is the smallest number divisible
by 10^5 and sum of digits as 5.

Input : N = 20
Output : 29900000000000000000000
```

Explanation

To make a number divisible by 10^N we need at least N zeros at the end of the number. To make the number smallest, we append exactly N zeros to the end of the number. Now, we need to ensure the sum of the digits is N. For this, we will try to make the length of the number as small as possible to get the answer. Thus we keep on inserting 9 into the number till the sum doesn't exceed N. If we have any remainder left, then we keep it as the first digit (most significant one) so that the resulting number is minimized.

The approach works well for all subtasks but there are 2 corner cases:

1. The first is that the final number may not fit into the data types present in C++/Java. Since we only need to output the number, we can use strings to store the answer.

2. The only corner case where the answer is 0 is N = 0.
3. There are no cases where the answer doesn't exist.

C++

```
 // CPP program to find smallest
// number to find smallest number
// with N as sum of digits and
// divisible by 10^N.
#include <bits/stdc++.h>
using namespace std;

void digitsNum(int N)
{
    // If N = 0 the string will be 0
    if (N == 0)
        cout << "0\n";

    // If n is not perfectly divisible
    // by 9 output the remainder
    if (N % 9 != 0)
        cout << (N % 9);

    // Print 9 N/9 times
    for (int i = 1; i <= (N / 9); ++i)
        cout << "9";

    // Append N zero's to the number so
    // as to make it divisible by 10^N
    for (int i = 1; i <= N; ++i)
        cout << "0";

    cout << "\n";
}

// Driver Code
int main()
{
    int N = 5;
    cout << "The number is : ";
    digitsNum(N);
    return 0;
}
```

Java

```
 // Java program to find smallest
// number to find smallest number
```

```
// with N as sum of digits and
// divisible by 10^N.
import java.io.*;

class GFG
{

static void digitsNum(int N)
{
    // If N = 0 the string will be 0
    if (N == 0)
    System.out.println("0");


    // If n is not perfectly divisible
    // by 9 output the remainder
    if (N % 9 != 0)
        System.out.print((N % 9));


    // Print 9 N/9 times
    for (int i = 1; i <= (N / 9); ++i)
        System.out.print("9");


    // Append N zero's to the number so
    // as to make it divisible by 10^N
    for (int i = 1; i <= N; ++i)
        System.out.print("0");
        System.out.print("" );

}

    // Driver Code
    public static void main (String[] args)
    {
    int N = 5;
    System.out.print("The number is : ");
    digitsNum(N);
    }
}

// This code is contributed by vt_m
```

Python3

```
 # Python program to find smallest
# number to find smallest number
```

```
# with N as sum of digits and
# divisible by 10^N.

import math
def digitsNum(N):

    # If N = 0 the string will be 0
    if (N == 0) :
        print("0", end = "")

    # If n is not perfectly divisible
    # by 9 output the remainder
    if (N % 9 != 0):
        print (N % 9, end ="")

    # Print 9 N/9 times
    for i in range( 1, int(N / 9) + 1) :
        print("9", end = "")

    # Append N zero's to the number so
    # as to make it divisible by 10^N
    for i in range(1, N + 1) :
        print("0", end = "")

    print()


# Driver Code
N = 5
print("The number is : ",end="")
digitsNum(N)

# This code is contributed by Gitanjali.
```

C#

```
 // C# program to find smallest
// number to find smallest number
// with N as sum of digits and
// divisible by 10^N.
using System;

class GFG
{

static void digitsNum(int N)
{
    // If N = 0 the string will be 0
```

```
    if (N == 0)
Console.Write("0");


    // If n is not perfectly divisible
    // by 9 output the remainder
    if (N % 9 != 0)
        Console.Write((N % 9));


    // Print 9 N/9 times
    for (int i = 1; i <= (N / 9); ++i)
        Console.Write("9");


    // Append N zero's to the number so
    // as to make it divisible by 10^N )
    for (int i = 1; i <= N; ++i)
        Console.Write("0");
        Console.WriteLine("" );

}

    // Driver Code
    public static void Main ()
    {
    int N = 5;
    Console.Write("The number is : ");
    digitsNum(N);
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find smallest
// number to find smallest number
// with N as sum of digits and
// divisible by 10^N.

function digitsNum($N)
{
    // If N = 0 the string will be 0
    if ($N == 0)
        echo "0\n";
```

```
    // If n is not perfectly divisible
    // by 9 output the remainder
    if ($N % 9 != 0)
        echo ($N % 9);

    // Print 9 N/9 times
    for ( $i = 1; $i <= ($N / 9); ++$i)
        echo "9";

    // Append N zero's to the number so
    // as to make it divisible by 10^N
    for ($i = 1; $i <= $N; ++$i)
        echo "0";

    echo "\n";
}

// Driver Code
$N = 5;
echo "The number is : ";
digitsNum($N);

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

Output :

```
The number is : 500000
```

Time Complexity : O(N)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/smallest-number-sum-digits-n-divisible-10n/

Chapter 135

Smallest subset with sum greater than all other elements

Smallest subset with sum greater than all other elements - GeeksforGeeks

Given an array of non-negative integers. Our task is to find minimum number of elements such that their sum should be greater than the sum of rest of the elements of the array.

Examples :

```
Input : arr[] = {3, 1, 7, 1}
Output : 1
Smallest subset is {7}. Sum of
this subset is greater than all
other elements {3, 1, 1}

Input : arr[] = {2, 1, 2}
Output : 2
In this example one element is not
enough. We can pick elements with
values 1, 2 or 2, 2. In any case,
the minimum count is 2.
```

The **Brute force** approach is to find sum of all the possible subsets and then compare sum with sum of remaining elements.

The **Efficient Approach** is to take the largest elements. We sort values in descending order, then take elements from the largest, until we get strictly more than half of total sum of the given array.
CPP

```
// CPP program to find minimum number of
```

```
// elements such that their sum is greater
// than sum of remaining elements of the array.
#include <bits/stdc++.h>
#include <string.h>
using namespace std;

// function to find minimum elements needed.
int minElements(int arr[], int n)
{
    // calculating HALF of array sum
    int halfSum = 0;
    for (int i = 0; i < n; i++)
        halfSum = halfSum + arr[i];
    halfSum = halfSum / 2;

    // sort the array in descending order.
    sort(arr, arr + n, greater<int>());

    int res = 0, curr_sum = 0;
    for (int i = 0; i < n; i++) {

        curr_sum += arr[i];
        res++;

        // current sum greater than sum
        if (curr_sum > halfSum)
            return res;
    }
    return res;
}

// Driver function
int main()
{
    int arr[] = {3, 1, 7, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << minElements(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java code to find minimum number of elements
// such that their sum is greater than sum of
// remaining elements of the array.
import java.io.*;
import java.util.*;
```

```
class GFG {

    // Function to find minimum elements needed
    static int minElements(int arr[], int n)
    {
        // Calculating HALF of array sum
        int halfSum = 0;
        for (int i = 0; i < n; i++)
            halfSum = halfSum + arr[i];
        halfSum = halfSum / 2;


        // Sort the array in ascending order and
        // start traversing array from the ascending
        // sort in descending order.
        Arrays.sort(arr);

        int res = 0, curr_sum = 0;
        for (int i = n-1; i >= 0; i--) {

            curr_sum += arr[i];
            res++;

            // Current sum greater than sum
            if (curr_sum > halfSum)
                return res;
        }
        return res;
    }

    // Driver Code
    public static void main (String[] args) {
        int arr[] = {3, 1, 7, 1};
        int n = arr.length;
        System.out.println(minElements(arr, n));
    }
    }

// This code is contributed by Gitanjali
```

Python3

```
 # Pyhton3 code to find minimum number of
# elements such that their sum is greater
# than sum of remaining elements of the array.

# function to find minimum elements needed.
def minElements(arr , n):
```

```
    # calculating HALF of array sum
    halfSum = 0
    for i in range(n):
        halfSum = halfSum + arr[i]

    halfSum = int(halfSum / 2)

    # sort the array in descending order.
    arr.sort(reverse = True)

    res = 0
    curr_sum = 0
    for i in range(n):

        curr_sum += arr[i]
        res += 1

        # current sum greater than sum
        if curr_sum > halfSum:
            return res

    return res

# driver code
arr = [3, 1, 7, 1]
n = len(arr)
print(minElements(arr, n) )

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

C#

```
 // C# code to find minimum number of elements
// such that their sum is greater than sum of
// remaining elements of the array.
using System;

class GFG {

    // Function to find minimum elements needed
    static int minElements(int []arr, int n)
    {

        // Calculating HALF of array sum
        int halfSum = 0;

        for (int i = 0; i < n; i++)
```

```
            halfSum = halfSum + arr[i];

        halfSum = halfSum / 2;

        // Sort the array in ascending order and
        // start traversing array from the ascending
        // sort in descending order.
        Array.Sort(arr);

        int res = 0, curr_sum = 0;
        for (int i = n-1; i >= 0; i--) {

            curr_sum += arr[i];
            res++;

            // Current sum greater than sum
            if (curr_sum > halfSum)
                return res;
        }

        return res;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {3, 1, 7, 1};
        int n = arr.Length;

        Console.WriteLine(minElements(arr, n));
    }
}

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

Output:

```
1
```

Time Complexity : O(n Log n)

Source

https://www.geeksforgeeks.org/smallest-subset-sum-greater-elements/

Chapter 136

Smallest sum contiguous subarray Set-2

Smallest sum contiguous subarray Set-2 - GeeksforGeeks

Given an array containing N integers. The task is to find the sum of the elements of the contiguous subarray having the smallest(minimum) sum.

Examples:

```
Input: arr[] = {3, -4, 2, -3, -1, 7, -5}
Output:-6

Input: arr = {2, 6, 8, 1, 4}
Output: 1
```

An approach has already been discussed in the previous post. In this post, a solution using the approach of Largest Sum Contiguous Subarray is discussed. This is based on the fact that in order to find the minimum contiguous sum we can first make the elements of the original array negative ie. Replace each ar[i] by -ar[i] and then apply Kadane Algorithm. Clearly, if this is the max sum formed then the minimum sum would be the negative of this sum.

Below is the implementation of above approach:
C++

```
 // C++ program for
// Smallest sum contiguous subarray | Set 2
#include <bits/stdc++.h>

using namespace std;
```

```
// function to find the smallest sum contiguous subarray
int smallestSumSubarr(int arr[], int n)
{
    // First invert the sign of the elements
    for (int i = 0; i < n; i++)
        arr[i] = -arr[i];

    // Apply the normal Kadane algorithm But on the elements
    // Of the array having inverted sign
    int sum_here = arr[0], max_sum = arr[0];

    for (int i = 1; i < n; i++) {

        sum_here = max(sum_here + arr[i], arr[i]);
        max_sum = max(max_sum, sum_here);
    }

    // Invert  the answer to get minimum val
    return (-1) * max_sum;
}

// Driver Code
int main()
{
    int arr[] = { 3, -4, 2, -3, -1, 7, -5 };

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

    cout << "Smallest sum: "
         << smallestSumSubarr(arr, n);
    return 0;
}
```

Java

```
 // Java program for Smallest
// sum contiguous subarray | Set 2
import java.io.*;

class GFG
{

// function to find the
// smallest sum contiguous
// subarray
static int smallestSumSubarr(int arr[],
                             int n)
{
```

```
    // First invert the
    // sign of the elements
    for (int i = 0; i < n; i++)
        arr[i] = -arr[i];

    // Apply the normal Kadane
    // algorithm But on the
    // elements Of the array
    // having inverted sign
    int sum_here = arr[0],
        max_sum = arr[0];

    for (int i = 1; i < n; i++)
    {
        sum_here = Math.max(sum_here +
                            arr[i], arr[i]);
        max_sum = Math.max(max_sum,
                           sum_here);
    }

    // Invert the answer
    // to get minimum val
    return (-1) * max_sum;
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {3, -4, 2, -3,
                -1, 7, -5};

    int n = arr.length;

    System.out.print("Smallest sum: " +
            smallestSumSubarr(arr, n));
}
}

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

C#

```
 // C# program for Smallest
// sum contiguous subarray | Set 2
using System;
class GFG
{
```

```
// function to find the
// smallest sum contiguous
// subarray
static int smallestSumSubarr(int []arr,
                             int n)
{
    // First invert the
    // sign of the elements
    for (int i = 0; i < n; i++)
        arr[i] = -arr[i];

    // Apply the normal Kadane
    // algorithm But on the
    // elements Of the array
    // having inverted sign
    int sum_here = arr[0],
        max_sum = arr[0];

    for (int i = 1; i < n; i++)
    {
        sum_here = Math.Max(sum_here +
                            arr[i], arr[i]);
        max_sum = Math.Max(max_sum,
                           sum_here);
    }

    // Invert the answer
    // to get minimum val
    return (-1) * max_sum;
}

// Driver Code
public static void Main ()
{
    int []arr = {3, -4, 2, -3,
                 -1, 7, -5};

    int n = arr.Length;

    Console.WriteLine("Smallest sum: " +
             smallestSumSubarr(arr, n));
}
}

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

Output:

```
Smallest sum: -6
```

Time Complexity: O(n)

Improved By : inderDuMCA

Source

https://www.geeksforgeeks.org/smallest-sum-contiguous-subarray-set-2/

Chapter 137

Sorting array with reverse around middle

Sorting array with reverse around middle - GeeksforGeeks

Consider the given array arr[], we need to find if we can sort array with given operation. We are only allowed to reverse subarray such that middle index (in case of odd elements) or indexes (2 indexes for even) are also middle index(s) of the subarray being reversed.

Examples:

```
Input : arr[] = {1, 6, 3, 4, 5, 2, 7}
Output : Yes
We can choose sub-array[3, 4, 5] on
reversing this we get [1, 6, 5, 4, 3, 2, 7]
again on selecting [6, 5, 4, 3, 2] and
reversing this one we get [1, 2, 3, 4, 5, 6, 7]
which is sorted at last thus it is possible
to sort on multiple reverse operation.

Input : arr[] = {1, 6, 3, 4, 5, 7, 2}
Output : No
```

One solution is we can rotate each element around the center, which gives two possibilities in the array i.e. the value at index 'i' or the value at index "length – 1 – i".
If array has n elements then 2^n combinations possible thus running time would be O(2^n).

Another solution can be make copy of the array and sort the copied array. Then compare each element of the sorted array with equivalent element of original array and its mirror image when pivot around center. Sorting the array takes O(n*logn) and 2n comparisons be required thus running time would be O(n*logn).

C++

```
 // CPP program to find possibility to sort
// by multiple subarray reverse operarion
#include <bits/stdc++.h>
using namespace std;

bool ifPossible(int arr[], int n)
{
    int cp[n];

    // making the copy of the original array
    copy(arr, arr + n, cp);

    // sorting the copied array
    sort(cp, cp + n);

    for (int i = 0; i < n; i++) {

        // checking mirror image of elements of sorted
        // copy array and equivalent element of original
        // array
        if (!(arr[i] == cp[i]) && !(arr[n - 1 - i] == cp[i]))
            return false;
    }

    return true;
}

// driver code
int main()
{
    int arr[] = { 1, 7, 6, 4, 5, 3, 2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (ifPossible(arr, n))
       cout << "Yes";
    else
       cout << "No";

    return 0;
}
```

Java

```
 // Java program to find possibility to sort
// by multiple subarray reverse operation
import java.util.*;
class GFG {

    static boolean ifPossible(int arr[], int n)
```

```
    {

        // making the copy of the original array
        int copy[] = Arrays.copyOf(arr, arr.length);

        // sorting the copied array
        Arrays.sort(copy);

        for (int i = 0; i < n; i++) {

            // checking mirror image of elements of
            // sorted copy array and equivalent element
            // of original array
            if (!(arr[i] == copy[i]) && !(arr[n - 1 - i] == copy[i]))
                return false;
        }

        return true;
    }

    // driver code
    public static void main(String[] args)
    {
        int arr[] = { 1, 7, 6, 4, 5, 3, 2, 8 };
        int n = arr.length;
        if (ifPossible(arr, n))
           System.out.println("Yes");
        else
           System.out.println("No");
    }
}
```

Python 3

```
 # Python 3 program to find
# possibility to sort by
# multiple subarray reverse
# operarion

def ifPossible(arr, n):

    cp = [0] * n

    # making the copy of
    # the original array
    cp = arr

    # sorting the copied array
```

```
    cp.sort()

    for i in range(0 , n) :

        # checking mirror image of
        # elements of sorted copy
        # array and equivalent element
        # of original array
        if (not(arr[i] == cp[i]) and not
               (arr[n - 1 - i] == cp[i])):
            return False

    return True

# Driver code
arr = [1, 7, 6, 4, 5, 3, 2, 8]
n = len(arr)
if (ifPossible(arr, n)):
    print("Yes")
else:
    print("No")

# This code is contributed by Smitha
```

C#

```
 // C# Program to answer queries on sum
// of sum of odd number digits of all
// the factors of a number
using System;

class GFG {

    static bool ifPossible(int []arr, int n)
    {
        int []cp = new int[n];

        // making the copy of the original
        // array
        Array.Copy(arr, cp, n);

        // sorting the copied array
        Array.Sort(cp);

        for (int i = 0; i < n; i++) {

            // checking mirror image of
            // elements of sorted copy
```

```
            // array and equivalent element
            // of original array
            if (!(arr[i] == cp[i]) &&
                 !(arr[n - 1 - i] == cp[i]))
                return false;
        }

        return true;
    }

    // Driver code
    public static void Main()
    {
        int []arr = new int[]{ 1, 7, 6, 4,
                               5, 3, 2, 8 };
        int n = arr.Length;

        if (ifPossible(arr, n))
            Console.WriteLine( "Yes");
        else
            Console.WriteLine( "No");
    }
}

// This code is contributed by Sam007
```

Output:

```
Yes
```

Improved By : Sam007, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/sorting-array-reverse-around-middle/

Chapter 138

Split n into maximum composite numbers

Split n into maximum composite numbers - GeeksforGeeks

Given n, print the maximum number of composite numbers that sum up to n. First few composite numbers are 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, ……..

Examples:

```
Input: 90
Output: 22
Explanation: If we add 21 4's, then we
get 84 and then add 6 to it, we get 90.

Input: 10
Output: 2
Explanation: 4 + 6 = 10
```

Below are some important observations.

1. If the number is less then 4, it won't have any combinations.
2. If the number is 5, 7, 11, it wont have any splitting.
3. Since smallest composite number is 4, it makes sense to use maximum number of 4s.
4. For numbers that don't leave a composite remainder when divided by 4, we do following. If remainder is 1, we subtract 9 from it to get the number which is perfectly divisible by 4. If the remainder is 2, then subtract 6 from it to make n a number which is perfectly divisible by 4. If the remainder is 3, then subtract 15 from it to make n perfectly divisible by 4, and 15 can be made up by 9 + 6.

So the main observation is to make n such that is composes of maximum no of 4's and the remaining can be made up by 6 and 9. We won't need composite numbers more then that, as the composite numbers above 9 can be made up of 4, 6, and 9.

Below is the implementation of the above approach

C++

```
 // CPP program to split a number into maximum
// number of composite numbers.
#include <bits/stdc++.h>
using namespace std;

// function to calculate the maximum number of
// composite numbers adding upto n
int count(int n)
{
    // 4 is the smallest composite number
    if (n < 4)
        return -1;

    // stores the remainder when n is divided
    // by 4
    int rem = n % 4;

    // if remainder is 0, then it is perfectly
    // divisible by 4.
    if (rem == 0)
        return n / 4;

    // if the remainder is 1
    if (rem == 1) {

        // If the number is less then 9, that
        // is 5, then it cannot be expressed as
        // 4 is the only composite number less
        // than 5
        if (n < 9)
            return -1;

        // If the number is greater then 8, and
        // has a remainder of 1, then express n
        // as n-9 a and it is perfectly divisible
        // by 4 and for 9, count 1.
        return (n - 9) / 4 + 1;
    }


    // When remainder is 2, just subtract 6 from n,
```

```
    // so that n is perfectly divisible by 4 and
    // count 1 for 6 which is subtracted.
    if (rem == 2)
        return (n - 6) / 4 + 1;


    // if the number is 7, 11 which cannot be
    // expressed as sum of any composite numbers
    if (rem == 3) {
        if (n < 15)
            return -1;

        // when the remainder is 3, then subtract
        // 15 from it and n becomes perfectly
        // divisible by 4 and we add 2 for 9 and 6,
        // which is getting subtracted to make n
        // perfectly divisible by 4.
        return (n - 15) / 4 + 2;
    }
}

// driver program to test the above function
int main()
{
    int n = 90;
    cout << count(n) << endl;

    n = 143;
    cout << count(n) << endl;
    return 0;
}
```

Java

```
 // Java program to split a number into maximum
// number of composite numbers.
import java.io.*;

class GFG
{
    // function to calculate the maximum number of
    // composite numbers adding upto n
    static int count(int n)
    {
        // 4 is the smallest composite number
        if (n < 4)
            return -1;
```

```
// stores the remainder when n is divided
// by 4
int rem = n % 4;

// if remainder is 0, then it is perfectly
// divisible by 4.
if (rem == 0)
    return n / 4;

// if the remainder is 1
if (rem == 1) {

    // If the number is less then 9, that
    // is 5, then it cannot be expressed as
    // 4 is the only composite number less
    // than 5
    if (n < 9)
        return -1;

    // If the number is greater then 8, and
    // has a remainder of 1, then express n
    // as n-9 a and it is perfectly divisible
    // by 4 and for 9, count 1.
    return (n - 9) / 4 + 1;
}


// When remainder is 2, just subtract 6 from n,
// so that n is perfectly divisible by 4 and
// count 1 for 6 which is subtracted.
if (rem == 2)
    return (n - 6) / 4 + 1;


// if the number is 7, 11 which cannot be
// expressed as sum of any composite numbers
if (rem == 3)
{
    if (n < 15)
        return -1;

    // when the remainder is 3, then subtract
    // 15 from it and n becomes perfectly
    // divisible by 4 and we add 2 for 9 and 6,
    // which is getting subtracted to make n
    // perfectly divisible by 4.
    return (n - 15) / 4 + 2;
}
```

```
        return 0;
    }

    // Driver program
    public static void main (String[] args)
    {
        int n = 90;
        System.out.println(count(n));

        n = 143;
        System.out.println(count(n));
    }
}

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

Python3

```
 # Python3 program to split a number into
# maximum number of composite numbers.

# Function to calculate the maximum number
# of composite numbers adding upto n
def count(n):

    # 4 is the smallest composite number
    if (n < 4):
        return -1

    # stores the remainder when n
    # is divided n is divided by 4
    rem = n % 4

    # if remainder is 0, then it is
    # perfectly divisible by 4.
    if (rem == 0):
        return n // 4

    # if the remainder is 1
    if (rem == 1):

        # If the number is less then 9, that
        # is 5, then it cannot be expressed as
        # 4 is the only composite number less
        # than 5
        if (n < 9):
            return -1
```

```
        # If the number is greater then 8, and
        # has a remainder of 1, then express n
        # as n-9 a and it is perfectly divisible
        # by 4 and for 9, count 1.
        return (n - 9) // 4 + 1



    # When remainder is 2, just subtract 6 from n,
    # so that n is perfectly divisible by 4 and
    # count 1 for 6 which is subtracted.
    if (rem == 2):
        return (n - 6) // 4 + 1


    # if the number is 7, 11 which cannot be
    # expressed as sum of any composite numbers
    if (rem == 3):
        if (n < 15):
            return -1

        # when the remainder is 3, then subtract
        # 15 from it and n becomes perfectly
        # divisible by 4 and we add 2 for 9 and 6,
        # which is getting subtracted to make n
        # perfectly divisible by 4.
        return (n - 15) // 4 + 2

# Driver Code
n = 90
print(count(n))

n = 143
print(count(n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to split a number into maximum
// number of composite numbers.
using System;

class GFG {

    // function to calculate the maximum number
    // of composite numbers adding upto n
    static int count(int n)
```

```
{

    // 4 is the smallest composite number
    if (n < 4)
        return -1;

    // stores the remainder when n is divided
    // by 4
    int rem = n % 4;

    // if remainder is 0, then it is perfectly
    // divisible by 4.
    if (rem == 0)
        return n / 4;

    // if the remainder is 1
    if (rem == 1) {

        // If the number is less then 9, that
        // is 5, then it cannot be expressed as
        // 4 is the only composite number less
        // than 5
        if (n < 9)
            return -1;

        // If the number is greater then 8, and
        // has a remainder of 1, then express n
        // as n-9 a and it is perfectly divisible
        // by 4 and for 9, count 1.
        return (n - 9) / 4 + 1;
    }


    // When remainder is 2, just subtract 6 from n,
    // so that n is perfectly divisible by 4 and
    // count 1 for 6 which is subtracted.
    if (rem == 2)
        return (n - 6) / 4 + 1;


    // if the number is 7, 11 which cannot be
    // expressed as sum of any composite numbers
    if (rem == 3)
    {
        if (n < 15)
            return -1;

        // when the remainder is 3, then subtract
```

```
            // 15 from it and n becomes perfectly
            // divisible by 4 and we add 2 for 9 and 6,
            // which is getting subtracted to make n
            // perfectly divisible by 4.
            return (n - 15) / 4 + 2;
        }

        return 0;
    }

    // Driver program
    public static void Main()
    {
        int n = 90;
        Console.WriteLine(count(n));

        n = 143;
        Console.WriteLine(count(n));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to split a number
// into maximum number of
// composite numbers.

// function to calculate the
// maximum number of composite
// numbers adding upto n
function c_ount($n)
{

    // 4 is the smallest
    // composite number
    if ($n < 4)
        return -1;

    // stores the remainder when
    // n is divided by 4
    $rem = $n % 4;

    // if remainder is 0, then it
    // is perfectly divisible by 4.
    if ($rem == 0)
```

```
    return $n / 4;

// if the remainder is 1
if ($rem == 1) {

    // If the number is less
    // then 9, that is 5, then
    // it cannot be expressed
    // as  4 is the only
    //composite number less
    // than 5
    if ($n < 9)
        return -1;

    // If the number is greater
    // then 8, and has a
    // remainder of 1, then
    // express n as n-9 a and
    // it is perfectly divisible
    // by 4 and for 9, count 1.
    return ($n - 9) / 4 + 1;
}


// When remainder is 2, just
// subtract 6 from n, so that n
// is perfectly divisible by 4
// and count 1 for 6 which is
// subtracted.
if ($rem == 2)
    return ($n - 6) / 4 + 1;


// if the number is 7, 11 which
// cannot be expressed as sum of
// any composite numbers
if ($rem == 3) {
    if ($n < 15)
        return -1;

    // when the remainder is 3,
    // then subtract 15 from it
    // and n becomes perfectly
    // divisible by 4 and we add
    // 2 for 9 and 6, which is
    // getting subtracted to make
    // n perfectly divisible by 4.
    return ($n - 15) / 4 + 2;
```

```
    }
}

// driver program to test the above
// function

    $n = 90;
    echo c_ount($n),"\n";

    $n = 143;
    echo c_ount($n);

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

Output:

```
22
34
```

Time complexity: O(1)
Auxiliary Space: O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/split-n-maximum-composite-numbers/

Chapter 139

Subarray whose absolute sum is closest to K

Subarray whose absolute sum is closest to K - GeeksforGeeks

Given an array of n elements and an integer K, the task is to find the subarray with minimum value of **a[i] + a[i + 1] + ... a[j] − K**. In other words, find the contiguous sub-array whose sum of elements shows minimum deviation from K or the subarray whose absolute sum is closest to K.

Example

> **Input:**: a[] = {1, 3, 7, 10}, K = 15
> **Output:** Subarray {7, 10}
> The contiguous sub-array [7, 10] shows minimum deviation of 2 from 15.
>
> **Input:** a[] = {1, 2, 3, 4, 5, 6}, K = 6
> **Output:** Subarray {1, 2, 3}
> The contiguous sub-array [1, 2, 3] shows minimum deviation of 0 from 6.

A **naive approach** would be to check if the sum of each contiguous sub-array and it's difference from K.

Below is the implementation of the above approach:

```
 # Python Code to find sub-array whose
# sum shows the minimum deviation

def getSubArray(arr, n, K):
    i = -1
    j = -1
    currSum = 0
    # starting index, ending index, Deviation
    result = [i, j, abs(K-abs(currSum))]
```

```
    # iterate i and j to get all subarrays
    for i in range(0, n):

        currSum = 0

        for j in range(i, n):
            currSum += arr[j]
            currDev = abs(K-abs(currSum))

            # found sub-array with less sum
            if (currDev < result[2]):
                result = [i, j, currDev]

            # exactly same sum
            if (currDev == 0):
                return result
    return result

# Driver Code
def main():
    arr = [15, -3, 5, 2, 7, 6, 34, -6]

    n = len(arr)

    K = 50

    [i, j, minDev] = getSubArray(arr, n, K)

    if(i ==-1):
        print("The empty array shows minimum Deviation")
        return 0

    for i in range(i, j + 1):
        print arr[i],


main()
```

Output:

```
-3 5 2 7 6 34
```

Time Complexity: O(N^2)

Efficient Approach: If the array only consists of **non-negative integers**, use the sliding window technique to improve the calculation time for sum in each iteration. The sliding

window technique reduces the complexity by calculating the new sub-array sum using the previous sub-array sum. Increase the right index till the difference (K-sum) is greater than zero. The first sub-array with negative (K-sum) is considered, and the next sub-array is with left index = i+1(where i is the current right index).

Below is the implementation of the above approach:

```
 # Python Code to find non-negative
# sub-array whose sum shows minimum deviation
# This works only if all elements
# in array are non-negative


# function to return the index
def getSubArray(arr, n, K):
    currSum = 0
    prevDif = 0
    currDif = 0
    result = [-1, -1, abs(K-abs(currSum))]
    resultTmp = result
    i = 0
    j = 0

    while(i<= j and j<n):

        # Add Last element tp currSum
        currSum += arr[j]

        # Save Difference of previous Iteration
        prevDif = currDif

        # Calculate new Difference
        currDif = K - abs(currSum)

        # When the Sum exceeds K
        if(currDif <= 0):
            if abs(currDif) < abs(prevDif):

            # Current Difference greater in magnitude
            # Store Temporary Result
                resultTmp = [i, j, currDif]
            else:

            # Diffence in Previous was lesser
            # In previous, Right index = j-1
                resultTmp = [i, j-1, prevDif]

            # In next iteration, Left Index Increases
            # but Right Index remains the Same
```

```
            # Update currSum and i Accordingly
            currSum -= (arr[i]+arr[j])

            i += 1

        # Case to simply increase Right Index
        else:
            resultTmp = [i, j, currDif]
            j += 1

        if(abs(resultTmp[2]) < abs(result[2])):
        # Check if lesser deviation found
            result = resultTmp

    return result

# Driver Code
def main():
    arr = [15, -3, 5, 2, 7, 6, 34, -6]

    n = len(arr)

    K = 50

    [i, j, minDev] = getSubArray(arr, n, K)

    if(i ==-1):
        print("The empty array shows minimum Deviation")
        return 0

    for i in range(i, j+1):
        print arr[i],


main()
```

Output:

```
-3 5 2 7 6 34
```

Time Complexity: O(N)

Source

https://www.geeksforgeeks.org/subarray-whose-absolute-sum-is-closest-to-k/

Chapter 140

Sum of Areas of Rectangles possible for an array

Sum of Areas of Rectangles possible for an array - GeeksforGeeks

Given an array, task is to compute the sum of all possible maximum area rectangle which can be formed from the array elements. Also, you can reduce the elements of the array by at most 1.

Examples :

```
Input :a = {10, 10, 10, 10, 11,
            10, 11, 10}
Output : 210
Explanation :
We can form two rectangles one square (10 * 10)
and one (11 * 10). Hence, total area = 100 + 110 = 210.

Input : a = { 3, 4, 5, 6 }
Output : 15
Explanation :
We can reduce 4 to 3 and 6 to 5 so that we got
rectangle of (3 * 5). Hence area = 15.

Input : a = { 3, 2, 5, 2 }
Output : 0
```

Naive Approach : Check for all possible four elements of the array and then whichever can form a rectangle. In these rectangles, separate all those rectangles which are of maximum area formed by these elements. After getting the rectangles and their areas, sum them all to get our desired solution.

Efficient Approach : To get the maximum area rectangle, first sort the elements of the array in non-increasing array. After sorting, start the procedure to select the elements of the array. Here, selection of two elements of array (as length of rectangle) is possible if elements of array are equal *(a[i] == a[i+1])* or if length of smaller element a[i+1] is one less than a[i] (*in this case we have our length a[i+1] because a[i] is decreased by 1*). One flag variable is maintained to check that *whether we get length and breadth both.* After getting length, set the flag variable, now calculate the breadth in the same way as we have done for length and sum the area of rectangle. After getting length and breadth both, again set the flag variable false so that we will now search for a new rectangle. This process is repeated and lastly, final sum of area is returned.

C++

```
 // CPP code to find sum of all
// area rectangle possible
#include <bits/stdc++.h>
using namespace std;

// Function to find
// area of rectangles
int MaxTotalRectangleArea(int a[],
                          int n)
{
    // sorting the array in
    // descending order
    sort(a, a + n, greater<int>());

    // store the final sum of
    // all the rectangles area
    // possible
    int sum = 0;
    bool flag = false;

    // temporary variable to store
    // the length of rectangle
    int len;

    for (int i = 0; i < n; i++)
    {

        // Selecting the length of
        // rectangle so that difference
        // between any two number is 1
        // only. Here length is selected
        // so flag is set
        if ((a[i] == a[i + 1] || a[i] -
            a[i + 1] == 1) && (!flag))
        {
            // flag is set means
```

```
            // we have got length of
            // rectangle
            flag = true;

            // length is set to
            // a[i+1] so that if
            // a[i] a[i+1] is less
            // than by 1 then also
            // we have the correct
            // choice for length
            len = a[i + 1];

            // incrementing the counter
            // one time more as we have
            // considered a[i+1] element
            // also so.
            i++;
        }

        // Selecting the width of rectangle
        // so that difference between any
        // two number is 1 only. Here width
        // is selected so now flag is again
        // unset for next rectangle
        else if ((a[i] == a[i + 1] ||
                a[i] - a[i + 1] == 1) && (flag))
        {
            // area is calculated for
            // rectangle
            sum = sum + a[i + 1] * len;

            // flag is set false
            // for another rectangle
            // which we can get from
            // elements in array
            flag = false;

            // incrementing the counter
            // one time more as we have
            // considered a[i+1] element
            // also so.
            i++;
        }
    }

    return sum;
}
```

```
// Driver code
int main()
{
    int a[] = { 10, 10, 10, 10,
                11, 10, 11, 10,
                9, 9, 8, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << MaxTotalRectangleArea(a, n);

    return 0;
}
```

Java

```
 // Java code to find sum of
// all area rectangle possible
import java.io.*;
import java.util.Arrays;

class GFG
{
    // Function to find
    // area of rectangles
    static int MaxTotalRectangleArea(int []a,
                                     int n)
    {

        // sorting the array in
        // descending order
        Arrays.sort(a);

        // store the final sum of
        // all the rectangles area
        // possible
        int sum = 0;
        boolean flag = false;

        // temporary variable to
        // store the length of rectangle
        int len = 0;

        for (int i = 0; i < n; i++)
        {

            // Selecting the length of
            // rectangle so that difference
            // between any two number is 1
```

```
// only. Here length is selected
// so flag is set
if ((a[i] == a[i + 1] ||
     a[i] - a[i + 1] == 1) &&
                    !flag)
{
    // flag is set means
    // we have got length of
    // rectangle
    flag = true;

    // length is set to
    // a[i+1] so that if
    // a[i] a[i+1] is less
    // than by 1 then also
    // we have the correct
    // choice for length
    len = a[i + 1];

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
          a[i] - a[i + 1] == 1) &&
                         (flag))
    {
        // area is calculated for
        // rectangle
        sum = sum + a[i + 1] * len;

        // flag is set false
        // for another rectangle
        // which we can get from
        // elements in array
        flag = false;

        // incrementing the counter
        // one time more as we have
        // considered a[i+1] element
```

```
                    // also so.
                    i++;
                }
        }

        return sum;
    }

    // Driver code
    public static void main (String args[])
    {
    int []a = { 10, 10, 10, 10,
                11, 10, 11, 10,
                9, 9, 8, 8 };
    int n = a.length;

    System.out.print(MaxTotalRectangleArea(a, n));
    }
}
// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python3 code to find sum
# of all area rectangle
# possible

# Function to find
# area of rectangles
def MaxTotalRectangleArea(a, n) :

    # sorting the array in
    # descending order
    a.sort(reverse = True)

    # store the final sum of
    # all the rectangles area
    # possible
    sum = 0
    flag = False

    # temporary variable to store
    # the length of rectangle
    len = 0
    i = 0

    while (i < n-1) :
```

```
if(i != 0) :
    i = i + 1

# Selecting the length of
# rectangle so that difference
# between any two number is 1
# only. Here length is selected
# so flag is set
if ((a[i] == a[i + 1] or
     a[i] - a[i + 1] == 1)
      and flag == False) :

    # flag is set means
    # we have got length of
    # rectangle
    flag = True

    # length is set to
    # a[i+1] so that if
    # a[i+1] is less than a[i]
    # by 1 then also we have
    # the correct chice for length
    len = a[i + 1]

    # incrementing the counter
    # one time more as we have
    # considered a[i+1] element
    # also so.
    i = i + 1

# Selecting the width of rectangle
# so that difference between any
# two number is 1 only. Here width
# is selected so now flag is again
# unset for next rectangle
elif ((a[i] == a[i + 1] or
      a[i] - a[i + 1] == 1)
        and flag == True) :

    # area is calculated for
    # rectangle
    sum = sum + a[i + 1] * len

    # flag is set false
    # for another rectangle
    # which we can get from
    # elements in array
    flag = False
```

```
            # incrementing the counter
            # one time more as we have
            # considered a[i+1] element
            # also so.
            i = i + 1

    return sum

# Driver code
a = [ 10, 10, 10, 10, 11, 10,
          11, 10, 9, 9, 8, 8 ]
n = len(a)

print (MaxTotalRectangleArea(a, n))

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# code to find sum of all area rectangle
// possible
using System;

class GFG {

    // Function to find
    // area of rectangles
    static int MaxTotalRectangleArea(int []a,
                                     int n)
    {

        // sorting the array in descending
        // order
        Array.Sort(a);

        // store the final sum of all the
        // rectangles area possible
        int sum = 0;
        bool flag = false;

        // temporary variable to store the
        // length of rectangle
        int len =0;

        for (int i = 0; i < n; i++)
        {
```

```
// Selecting the length of
// rectangle so that difference
// between any two number is 1
// only. Here length is selected
// so flag is set
if ((a[i] == a[i + 1] || a[i] -
    a[i + 1] == 1) && (!flag))
{
    // flag is set means
    // we have got length of
    // rectangle
    flag = true;

    // length is set to
    // a[i+1] so that if
    // a[i] a[i+1] is less
    // than by 1 then also
    // we have the correct
    // choice for length
    len = a[i + 1];

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
        a[i] - a[i + 1] == 1) && (flag))
    {
        // area is calculated for
        // rectangle
        sum = sum + a[i + 1] * len;

        // flag is set false
        // for another rectangle
        // which we can get from
        // elements in array
        flag = false;

        // incrementing the counter
```

```
                    // one time more as we have
                    // considered a[i+1] element
                    // also so.
                    i++;
                }
        }

        return sum;
    }

    // Driver code
    static public void Main ()
    {
            int []a = { 10, 10, 10, 10,
                        11, 10, 11, 10,
                        9, 9, 8, 8 };
    int n = a.Length;

    Console.WriteLine(
                MaxTotalRectangleArea(a, n));
    }
}

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

PHP

```
 <?php
// PHP code to find sum
// of all area rectangle
// possible

// Function to find
// area of rectangles
function MaxTotalRectangleArea( $a, $n)
{
    // sorting the array in
    // descending order
    rsort($a);

    // store the final sum of
    // all the rectangles area
    // possible
    $sum = 0;
    $flag = false;

    // temporary variable to store
    // the length of rectangle
```

```
$len;

for ( $i = 0; $i < $n; $i++)
{

    // Selecting the length of
    // rectangle so that difference
    // between any two number is 1
    // only. Here length is selected
    // so flag is set
    if (($a[$i] == $a[$i + 1] or $a[$i] -
         $a[$i + 1] == 1) and (!$flag))
    {
        // flag is set means
        // we have got length of
        // rectangle
        $flag = true;

        // length is set to
        // a[i+1] so that if
        // a[i+1] is less than a[i]
        // by 1 then also we have
        // the correct chice for length
        $len = $a[$i + 1];

        // incrementing the counter
        // one time more as we have
        // considered a[i+1] element
        // also so.
        $i++;
    }

    // Selecting the width of rectangle
    // so that difference between any
    // two number is 1 only. Here width
    // is selected so now flag is again
    // unset for next rectangle
    else if (($a[$i] == $a[$i + 1] or
              $a[$i] - $a[$i + 1] == 1) and
             ($flag))
        {
            // area is calculated for
            // rectangle
            $sum = $sum + $a[$i + 1] * $len;

            // flag is set false
            // for another rectangle
            // which we can get from
```

```
                // elements in array
                $flag = false;

                // incrementing the counter
                // one time more as we have
                // considered a[i+1] element
                // also so.
                $i++;
            }
    }

    return $sum;
}

// Driver code
$a = array( 10, 10, 10, 10,
            11, 10, 11, 10,
            9, 9, 8, 8 );
$n = count($a);

echo MaxTotalRectangleArea($a, $n);

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

Output :

```
282
```

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

Improved By : vt_m, manishshaw1

Source

https://www.geeksforgeeks.org/sum-area-rectangles-possible-array/

Chapter 141

Sum of minimum difference between consecutive elements of an array

Sum of minimum difference between consecutive elements of an array - GeeksforGeeks

Given an array of pairs where each pair represents a range, the task is to find the sum of the minimum difference between the consecutive elements of an array where the array is filled in the below manner:

- Each element of an array lies in the range given at its corresponding index in the range array.
- Final sum of difference of consecutive elements in the array formed is minimum.

Examples:

> **Input:** range[] = {{2, 4}, {3, 6}, {1, 5}, {1, 3}, {2, 7}}
> **Output:** 0
> The result is 0 because the array {3, 3, 3, 3, 3} is choosen
> then the sum of difference of consecutive element will be
> { 3-3 + 3-3 + 3-3 + 3-3 } = 0 which the minimum.
>
> **Input:** range[] = {{1, 3}, {2, 5}, {6, 8}, {1, 2}, {2, 3}}
> **Output:** 7
> The result is 7 because if the array {3, 3, 6, 2, 2} is choosen
> then the sum of difference of consecutive element will be
> { 3-3 + 6-3 + 2-6 + 2-2 }= 7 which is the minimum.

Approach: A greedy approach has been followed to solve the above problem. Initially we have to fill the array in such a way that the sum of the difference obtained is minimum. The greedy approach is as follows:

- If the range of the previous index intersects the range of current index then, in this case, the minimum difference will be 0 and store the highest value and the lowest value of intersecting range.
- If the lowest value of the range of the previous index is greater then the highest value for the current index than in this case the least possible sum is the difference in the lowest value of the previous range and the highest value stored of the current range.
- If the highest range of the previous index is lower than the lowest value of current range then the minimum sum is the difference in the lowest value stored for the current index and the highest range of the previous index.

Below is the implementation of the above approach:

C++

```
 // C++ program for finding the minimum sum of
// difference between consecutive elements
#include <bits/stdc++.h>
using namespace std;

// function to find minimum sum of
// difference of consecutive element
int solve(pair<int, int> v[], int n)
{
    // ul to store upper limit
    // ll to store lower limit
    int ans, ul, ll;

    // storethe lower range in ll
    // and upper range in ul
    ll = v[0].first;
    ul = v[0].second;

    // inititalize the answer with 0
    ans = 0;

    // iterate for all ranges
    for (int i = 1; i < n; i++) {

        // case 1, in this case the difference will be 0
        if ((v[i].first <= ul && v[i].first >= ll) ||
           (v[i].second >= ll && v[i].second <= ul)) {

            // change upper limit and lower limit
            if (v[i].first > ll) {
                ll = v[i].first;
            }
            if (v[i].second < ul) {
```

```
                ul = v[i].second;
            }
        }

        // case 2
        else if (v[i].first > ul) {

            // store the difference
            ans += abs(ul - v[i].first);
            ul = v[i].first;
            ll = v[i].first;
        }

        // case 3
        else if (v[i].second < ll) {

            // store the difference
            ans += abs(ll - v[i].second);
            ul = v[i].second;
            ll = v[i].second;
        }
    }
    return ans;
}
// Driver code
int main()
{
    // array of range

    pair<int, int> v[] = { { 1, 3 }, { 2, 5 },
                { 6, 8 }, { 1, 2 }, { 2, 3 } };
    int n = sizeof(v) / sizeof(v[0]);
    cout << solve(v, n) << endl;

    return 0;
}
```

Java

```
 // Java program for finding the
// minimum sum of difference
// between consecutive elements
import java.io.*;

class GFG
{

// function to find minimum
```

```
// sum of difference of
// consecutive element
static int solve(int[][] v, int n)
{
    // ul to store upper limit
    // ll to store lower limit
    int ans, ul, ll;
    int first = 0;
    int second = 1;

    // storethe lower range
    // in ll and upper range
    // in ul
    ll = v[0][first];
    ul = v[0][second];

    // inititalize the
    // answer with 0
    ans = 0;

    // iterate for all ranges
    for (int i = 1; i < n; i++)
    {

        // case 1, in this case
        // the difference will be 0
        if ((v[i][first] <= ul &&
              v[i][first] >= ll) ||
            (v[i][second] >= ll &&
             v[i][second] <= ul))
        {

            // change upper limit
            // and lower limit
            if (v[i][first] > ll)
            {
                ll = v[i][first];
            }
            if (v[i][second] < ul)
            {
                ul = v[i][second];
            }
        }

        // case 2
        else if (v[i][first] > ul)
        {
```

```
            // store the difference
            ans += Math.abs(ul - v[i][first]);
            ul = v[i][first];
            ll = v[i][first];
        }

        // case 3
        else if (v[i][second] < ll)
        {

            // store the difference
            ans += Math.abs(ll - v[i][second]);
            ul = v[i][second];
            ll = v[i][second];
        }
    }
    return ans;
}

// Driver code
public static void main(String []args)
{
    // array of range

    int[][] v = {{ 1, 3 }, { 2, 5 },
                 { 6, 8 }, { 1, 2 },
                 { 2, 3 }};
    int n = 5;
    System.out.println(solve(v, n));
}
}

// This code is contributed
// by chandan_jnu
```

Python

```
 # Python program for finding
# the minimum sum of difference
# between consecutive elements

class pair:
    first = 0
    second = 0

    def __init__(self, a, b):
        self.first = a
        self.second = b
```

```
# function to find minimum
# sum of difference of
# consecutive element
def solve(v, n):

    # ul to store upper limit
    # ll to store lower limit
    ans = 0; ul = 0; ll = 0;

    # storethe lower range
    # in ll and upper range
    # in ul
    ll = v[0].first
    ul = v[0].second

    # inititalize the
    # answer with 0
    ans = 0

    # iterate for all ranges
    for i in range(1, n):

        # case 1, in this case
        # the difference will be 0
        if (v[i].first <= ul and
            v[i].first >= ll) or \
           (v[i].second >= ll and
            v[i].second <= ul):

            # change upper limit
            # and lower limit
            if v[i].first > ll:
                ll = v[i].first

            if v[i].second < ul:
                ul = v[i].second;

        # case 2
        elif v[i].first > ul:

            # store the difference
            ans += abs(ul - v[i].first)
            ul = v[i].first
            ll = v[i].first

        # case 3
        elif v[i].second < ll:
```

```
            # store the difference
            ans += abs(ll - v[i].second);
            ul = v[i].second;
            ll = v[i].second;
    return ans

# Driver code

# array of range
v = [pair(1, 3), pair(2, 5),
     pair(6, 8), pair(1, 2),
                 pair(2, 3) ]
n = len(v)
print(solve(v, n))

# This code is contributed
# by Harshit Saini
```

C#

```
 // C# program for finding the
// minimum sum of difference
// between consecutive elements
using System;

class GFG
{
// function to find minimum
// sum of difference of
// consecutive element
static int solve(int[,] v, int n)
{
    // ul to store upper limit
    // ll to store lower limit
    int ans, ul, ll;
    int first = 0;
    int second = 1;

    // storethe lower range
    // in ll and upper range
    // in ul
    ll = v[0, first];
    ul = v[0, second];

    // inititalize the
    // answer with 0
    ans = 0;
```

```
    // iterate for all ranges
    for (int i = 1; i < n; i++)
    {

        // case 1, in this case
        // the difference will be 0
        if ((v[i, first] <= ul &&
             v[i, first] >= ll) ||
            (v[i, second] >= ll &&
             v[i, second] <= ul))
        {

            // change upper limit
            // and lower limit
            if (v[i, first] > ll)
            {
                ll = v[i, first];
            }
            if (v[i, second] < ul)
            {
                ul = v[i, second];
            }
        }

        // case 2
        else if (v[i, first] > ul)
        {

            // store the difference
            ans += Math.Abs(ul - v[i, first]);
            ul = v[i, first];
            ll = v[i, first];
        }

        // case 3
        else if (v[i, second] < ll)
        {

            // store the difference
            ans += Math.Abs(ll - v[i, second]);
            ul = v[i, second];
            ll = v[i, second];
        }
    }
    return ans;
}
```

```
// Driver code
static void Main()
{
    // array of range

    int[,] v = new int[5,2]{ { 1, 3 }, { 2, 5 },
                             { 6, 8 }, { 1, 2 },
                             { 2, 3 } };
    int n = 5;
    Console.WriteLine(solve(v, n));
}
}

// This code is contributed
// by chandan_jnu
```

PHP

```
 <?php
// PHP program for finding the
// minimum sum of difference
// between consecutive elements

// function to find minimum
// sum of difference of
// consecutive element
function solve($v, $n)
{
// ul to store upper limit
// ll to store lower limit
$ans; $ul; $ll;
$first = 0;
$second = 1;

// storethe lower range
// in ll and upper range
// in ul
$ll = $v[0][$first];
$ul = $v[0][$second];

// inititalize the
// answer with 0
$ans = 0;

// iterate for all ranges
for ($i = 1; $i <$n; $i++)
{
```

```
    // case 1, in this case
    // the difference will be 0
    if (($v[$i][$first] <= $ul and
         $v[$i][$first] >= $ll) or
        ($v[$i][$second] >= $ll and
         $v[$i][$second] <= $ul))
    {

        // change upper limit
        // and lower limit
        if ($v[$i][$first] > $ll)
        {
            $ll = $v[$i][$first];
        }
        if ($v[$i][$second] < $ul)
        {
            $ul = $v[$i][$second];
        }
    }

    // case 2
    else if ($v[$i][$first] > $ul)
    {

        // store the difference
        $ans += abs($ul - $v[$i][$first]);
        $ul = $v[$i][$first];
        $ll = $v[$i][$first];
    }

    // case 3
    else if ($v[$i][$second] < $ll)
    {

        // store the difference
        $ans += abs($ll - $v[$i][$second]);
        $ul = $v[$i][$second];
        $ll = $v[$i][$second];
    }
}
return $ans;
}

// Driver code

// array of range
$v = array(array( 1, 3 ),
```

```
            array( 2, 5 ),
            array( 6, 8 ),
            array( 1, 2 ),
            array( 2, 3 ));
$n = 5;
echo(solve($v, $n));

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

Output:

```
7
```

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

Improved By : Harshit Saini, Chandan_Kumar

Source

https://www.geeksforgeeks.org/sum-of-minimum-difference-between-consecutive-elements-of-an-array/

Chapter 142

Value in a given range with maximum XOR

Value in a given range with maximum XOR - GeeksforGeeks

Given positive integers N, L, and R, we have to find the maximum value of N X, where X [L, R].

Examples:

> **Input :** N = 7
> L = 2
> R = 23
> **Output :** 23
> Explanation : When X = 16, we get 7 16 = 23 which is the maximum value for all X [2, 23].
>
> **Input :** N = 10
> L = 5
> R = 12
> **Output :** 15
> Explanation : When X = 5, we get 10 5 = 15 which is the maximum value for all X [5, 12].

Brute force approach: We can solve this problem using brute force approach by looping over all integers over the range [L, R] and taking their XOR with N, while keeping a record of the maximum result encountered so far. The complexity of this algorithm will be O(R – L), and it is not feasible when the input variables approach high values such as 10^9.

Efficient approach: Since the XOR of two bits is 1 if and only if they are complementary to each other, we need X to have complementary bits to that of N to have the maximum value. We will iterate from the largest bit ($\log_2(R)^{th}$ bit) to the lowest (0^{th} bit). The following two cases can arise for each bit:

1. If the bit is not set, i.e. 0, we will try to set it in X. If setting this bit to 1 results in X exceeding R, then we will not set it.
2. If the bit is set, i.e. 1, then we will try to unset it in X. If the current value of X is already greater than or equal to L, then we can safely unset the bit. In the other case, we will check if setting all of the next sits is enough to keep X >= L. If not, then we are required to set the current bit. Observe that setting all of the next bits is equivalent to adding $(1 << b) - 1$, where b is the current bit.

The time complexity of this approach is $O(\log_2(R))$.

C++

```
 // CPP program to find the x in range [l, r]
// such that x ^ n is maximum.
#include <cmath>
#include <iostream>
using namespace std;

// Function to calculate the maximum value of
// N ^ X, where X is in the range [L, R]
int maximumXOR(int n, int l, int r)
{
    int x = 0;
    for (int i = log2(r); i >= 0; --i) {
        if (n & (1 << i)) { // Set bit
            if ((x > r) || (x + (1 << i) - 1 < l))
                x ^= (1 << i);
        }
        else { // Unset bit
            if ((x ^ (1 << i)) <= r)
                x ^= (1 << i);
        }
    }
    return n ^ x;
}

// Driver function
int main()
{
    int n = 7, l = 2, r = 23;
    cout << "The output is " << maximumXOR(n, l, r);
    return 0;
}
```

Java

```
 // Java program to find the x in range [l, r]
```

```
// such that x ^ n is maximum.

import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
// Function to calculate the maximum value of
// N ^ X, where X is in the range [L, R]
static int maximumXOR(int n, int l, int r)
{
    int x = 0;
    for (int i = (int)(Math.log(r)/Math.log(2)); i >= 0; --i)
    {
        if ((n & (1 << i))>0) // Set bit
        {
            if ((x > r) || (x + (1 << i) - 1 < l))
                x ^= (1 << i);
        }
        else // Unset bit
        {
            if ((x ^ (1 << i)) <= r)
                x ^= (1 << i);
        }
    }
    return n ^ x;
}

// Driver function
public static void main(String args[])
{
    int n = 7, l = 2, r = 23;
    System.out.println( "The output is " + maximumXOR(n, l, r));

}
}

// This code is Contributed by tufan_gupta2000
```

Output:

```
The output is 23
```

Improved By : tufan_gupta2000

Source

https://www.geeksforgeeks.org/value-in-a-given-range-with-maximum-xor/

Chapter 143

Water Connection Problem

Water Connection Problem - GeeksforGeeks

Every house in the colony has at most one pipe going into it and at most one pipe going out of it. Tanks and taps are to be installed in a manner such that every house with one outgoing pipe but no incoming pipe gets a tank installed on its roof and every house with only an incoming pipe and no outgoing pipe gets a tap.'
Given two integers **n** and p denoting the number of houses and the number of pipes. The connections of pipe among the houses contain three input values: **a__i**, **b__i**, **d__i** denoting the pipe of diameter **d__i** from house **a__i** to house **b__i**, find out the efficient solution for the network.
The output will contain the number of pairs of tanks and taps **t** installed in first line and the next **t** lines contain three integers: house number of tank, house number of tap and the minimum diameter of pipe between them.

Examples:

```
Input : 4 2
        1 2 60
        3 4 50
Output :2
        1 2 60
        3 4 50
Explanation:
Connected components are:
1->2 and 3->4
Therefore, our answer is 2 followed by
1 2 60 and 3 4 50.

Input :9 6
       7 4 98
       5 9 72
       4 6 10
```

```
       2 8 22
       9 7 17
       3 1 66
Output :3
        2 8 22
        3 1 66
        5 6 10
Explanation:
Connected components are 3->1,
5->9->7->4->6 and 2->8.
Therefore, our answer is 3 followed by
2 8 22, 3 1 66, 5 6 10
```

Approach:
Perform DFS from appropriate houses to find all different connected components. The number of different connected components is our answer t.
The next t lines of the output are the beginning of the connected component, end of the connected component and the minimum diameter from the start to the end of the connected component in each line.
Since, tanks can be installed only on the houses having outgoing pipe and no incoming pipe, therefore these are appropriate houses to start DFS from i.e. perform DFS from such unvisited houses.

Below is the implementation of above approach:
C++

```
 // C++ program to find efficient
// solution for the network
#include <bits/stdc++.h>
using namespace std;

// number of houses and number
// of pipes
int n, p;

// Array rd stores the
// ending vertex of pipe
int rd[1100];

// Array wd stores the value
// of diameters between two pipes
int wt[1100];

// Array cd stores the
// starting end of pipe
int cd[1100];

// Vector a, b, c are used
```

```
// to store the final output
vector<int> a;
vector<int> b;
vector<int> c;

int ans;

int dfs(int w)
{
    if (cd[w] == 0)
        return w;
    if (wt[w] < ans)
        ans = wt[w];
    return dfs(cd[w]);
}

// Function performing calculations.
void solve(int arr[][3])
{
    int i = 0;

    while (i < p) {

        int q = arr[i][0], h = arr[i][1],
            t = arr[i][2];

        cd[q] = h;
        wt[q] = t;
        rd[h] = q;
        i++;
    }

    a.clear();
    b.clear();
    c.clear();

    for (int j = 1; j <= n; ++j)

        /*If a pipe has no ending vertex
        but has starting vertex i.e is
        an outgoing pipe then we need
        to start DFS with this vertex.*/
        if (rd[j] == 0 && cd[j]) {
            ans = 1000000000;
            int w = dfs(j);

            // We put the details of component
            // in final output array
```

```
            a.push_back(j);
            b.push_back(w);
            c.push_back(ans);
        }

    cout << a.size() << endl;
    for (int j = 0; j < a.size(); ++j)
        cout << a[j] << " " << b[j]
             << " " << c[j] << endl;
}

// driver function
int main()
{
    n = 9, p = 6;

    memset(rd, 0, sizeof(rd));
    memset(cd, 0, sizeof(cd));
    memset(wt, 0, sizeof(wt));

    int arr[][3] = { { 7, 4, 98 },
                     { 5, 9, 72 },
                     { 4, 6, 10 },
                     { 2, 8, 22 },
                     { 9, 7, 17 },
                     { 3, 1, 66 } };

    solve(arr);
    return 0;
}
```

Java

```
 // Java program to find efficient
// solution for the network
import java.util.*;

class GFG {

    // number of houses and number
    // of pipes
    static int n, p;

    // Array rd stores the
    // ending vertex of pipe
    static int rd[] = new int[1100];

    // Array wd stores the value
```

```
// of diameters between two pipes
static int wt[] = new int[1100];

// Array cd stores the
// starting end of pipe
static int cd[] = new int[1100];

// arraylist a, b, c are used
// to store the final output
static List <Integer> a =
          new ArrayList<Integer>();

static List <Integer> b =
          new ArrayList<Integer>();

static List <Integer> c =
          new ArrayList<Integer>();

static int ans;

static int dfs(int w)
{
    if (cd[w] == 0)
        return w;
    if (wt[w] < ans)
        ans = wt[w];

    return dfs(cd[w]);
}

// Function to perform calculations.
static void solve(int arr[][])
{
    int i = 0;

    while (i < p)
    {

        int q = arr[i][0];
        int h = arr[i][1];
        int t = arr[i][2];

        cd[q] = h;
        wt[q] = t;
        rd[h] = q;
        i++;
    }
```

```
    a=new ArrayList<Integer>();
    b=new ArrayList<Integer>();
    c=new ArrayList<Integer>();

    for (int j = 1; j <= n; ++j)

        /*If a pipe has no ending vertex
        but has starting vertex i.e is
        an outgoing pipe then we need
        to start DFS with this vertex.*/
        if (rd[j] == 0 && cd[j]>0) {
            ans = 1000000000;
            int w = dfs(j);

            // We put the details of
            // component in final output
            // array
            a.add(j);
            b.add(w);
            c.add(ans);
        }

    System.out.println(a.size());

    for (int j = 0; j < a.size(); ++j)
        System.out.println(a.get(j) + " "
            + b.get(j) + " " + c.get(j));
}

// main function
public static void main(String args[])
{
    n = 9;
    p = 6;

    // set the value of the araray
    // to zero
    for(int i = 0; i < 1100; i++)
        rd[i] = cd[i] = wt[i] = 0;

    int arr[][] = { { 7, 4, 98 },
                    { 5, 9, 72 },
                    { 4, 6, 10 },
                    { 2, 8, 22 },
                    { 9, 7, 17 },
                    { 3, 1, 66 } };
    solve(arr);
}
```

```
}

// This code is contributed by Arnab Kundu
```

Output:

```
3
2 8 22
3 1 66
5 6 10
```

Improved By : andrew1234, R0SHANRK_1995

Source

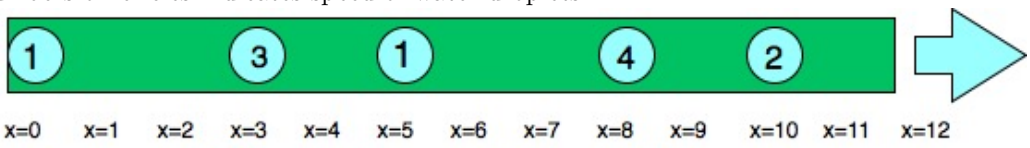
https://www.geeksforgeeks.org/water-connection-problem/

Chapter 144

Water drop problem

Water drop problem - GeeksforGeeks

Consider a pipe of length L. The pipe has N water droplets at N different positions within it. Each water droplet is moving towards the end of the pipe(x=L) at different rates. When a water droplet mixes with another water droplet, it assumes the speed of the water droplet it is mixing with. Determine the no of droplets that come out of the end of the pipe.
Refer to the figure below:
Numbers on circles indicates speed of water droplets



Examples:

```
Input: length = 12, position = [10, 8, 0, 5, 3],
       speed = [2, 4, 1, 1, 3]
Output: 3
Explanation:
Droplets starting at x=10 and x=8 become a droplet,
meeting each other at x=12 at time =1 sec.
The droplet starting at 0 doesn't mix with any
other droplet, so it is a drop by itself.
Droplets starting at x=5 and x=3 become a single
drop, mixing with each other at x=6 at time = 1 sec.
Note that no other droplets meet these drops before
the end of the pipe, so the answer is 3.
Refer to the figure below
Numbers on circles indicates speed of water droplets.
```

Approach:
This problem uses greedy technique.

A drop will mix with another drop if two conditions are met:
1. If the drop is faster than the drop it is mixing with
2. If the position of the faster drop is behind the slower drop.
We use an array of pairs to store the position and the time that ith drop would take to reach the end of the pipe. Then we sort the array according to the position of the drops. Now we have a fair idea of which drops lie behind which drops and their respective time taken to reach the end.More time means less speed and less time means more speed. Now all the drops before a slower drop will mix with it. And all the drops after the slower drop with mix with the next slower drop and so on.
For example if the times to reach the end are- 12, 3, 7, 8, 1 (sorted according to positions)
0th drop is slowest, it won't mix with the next drop
1st drop is faster than the 2nd drop so they will mix and 2nd drop is faster than 3rd drop so all three will mix together. They cannot mix with the 4th drop because that is faster.
So we use a stack to maintain the local maxima of the times.
No of local maximal + residue(drops after last local maxima) = total no of drops

```
 #include <bits/stdc++.h>
using namespace std;
int drops(int length, int position[], int speed[], int n)
{
    // stores position and time taken by a single
    // drop to reach the end as a pair
    vector<pair<int, double> > m(n);

    int i;
    for (i = 0; i < n; i++) {

        // calculates distance needs to be
        // covered by the ith drop
        int p = length - position[i];

        // inserts initial position of the
        // ith drop to the pair
        m[i].first = position[i];

        // inserts time taken by ith drop to reach
        // the end to the pair
        m[i].second = p * 1.0 / speed[i];
    }

    // sorts the pair according to increasing
    // order of their positions
    sort(m.begin(), m.end());
    int k = 0; // counter for no of final drops

    // stack to maintain the next slower drop
    // which might coalesce with the current drop
    stack<double> s;
```

```
    // we traverse the array demo right to left
    // to determine the slower drop
    for (i = n - 1; i >= 0; i--)
    {
        if (s.empty()) {
            s.push(m[i].second);
        }

        // checks for next slower drop
        if (m[i].second > s.top())
        {
            s.pop();
            k++;
            s.push(m[i].second);
        }
    }

    // calculating residual drops in the pipe
    if (!s.empty())
    {
        s.pop();
        k++;
    }
    return k;
}

// driver function
int main()
{
    int length = 12; // length of pipe
    int position[] = { 10, 8, 0, 5, 3 }; // position of droplets
    int speed[] = { 2, 4, 1, 1, 3 }; // speed of each droplets
    int n = sizeof(speed)/sizeof(speed[0]);
    cout << drops(length, position, speed, n);
    return 0;
}
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/water-drop-problem/

Chapter 145

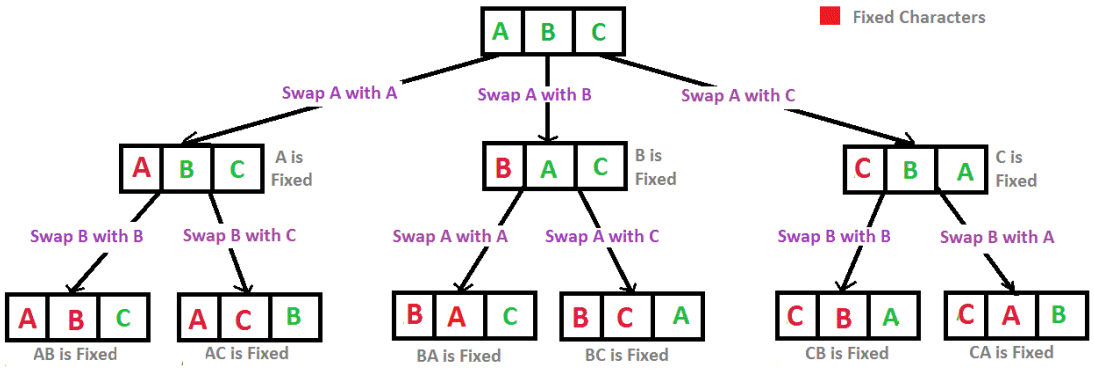
Write a program to print all permutations of a given string

Write a program to print all permutations of a given string - GeeksforGeeks

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

Here is a solution that is used as a basis in backtracking.



Recursion Tree for Permutations of String "ABC"

C/C++

```
 // 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;
}

/* 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;
}
```

Java

```
 // Java program to print all permutations of a
// given string.
public class Permutation
{
    public static void main(String[] args)
```

```
    {
        String str = "ABC";
        int n = str.length();
        Permutation permutation = new Permutation();
        permutation.permute(str, 0, n-1);
    }

    /**
     * permutation function
     * @param str string to calculate permutation for
     * @param l starting index
     * @param r end index
     */
    private void permute(String str, int l, int r)
    {
        if (l == r)
            System.out.println(str);
        else
        {
            for (int i = l; i <= r; i++)
            {
                str = swap(str,l,i);
                permute(str, l+1, r);
                str = swap(str,l,i);
            }
        }
    }

    /**
     * Swap Characters at position
     * @param a string value
     * @param i position 1
     * @param j position 2
     * @return swapped string
     */
    public String swap(String a, int i, int j)
    {
        char temp;
        char[] charArray = a.toCharArray();
        temp = charArray[i] ;
        charArray[i] = charArray[j];
        charArray[j] = temp;
        return String.valueOf(charArray);
    }

}

// This code is contributed by Mihir Joshi
```

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
```

C#

```
 // C# program to print all
// permutations of a given string.
using System;

class GFG
{
    /**
    * permutation function
    * @param str string to
       calculate permutation for
    * @param l starting index
    * @param r end index
    */
    private static void permute(String str,
                                int l, int r)
```

```
    {
        if (l == r)
            Console.WriteLine(str);
        else
        {
            for (int i = l; i <= r; i++)
            {
                str = swap(str, l, i);
                permute(str, l + 1, r);
                str = swap(str, l, i);
            }
        }
    }

    /**
    * Swap Characters at position
    * @param a string value
    * @param i position 1
    * @param j position 2
    * @return swapped string
    */
    public static String swap(String a,
                              int i, int j)
    {
        char temp;
        char[] charArray = a.ToCharArray();
        temp = charArray[i] ;
        charArray[i] = charArray[j];
        charArray[j] = temp;
        string s = new string(charArray);
        return s;
    }

// Driver Code
public static void Main()
{
    String str = "ABC";
    int n = str.Length;
    permute(str, 0, n-1);
}
}

// This code is contributed by mits
```

PHP

```
 <?php
// PHP program to print all
```

```
// permutations of a given string.


/**
* permutation function
* @param str string to
*  calculate permutation for
* @param l starting index
* @param r end index
*/
function permute($str, $l, $r)
{
    if ($l == $r)
        echo $str. "\n";
    else
    {
        for ($i = $l; $i <= $r; $i++)
        {
            $str = swap($str, $l, $i);
            permute($str, $l + 1, $r);
            $str = swap($str, $l, $i);
        }
    }
}

/**
* Swap Characters at position
* @param a string value
* @param i position 1
* @param j position 2
* @return swapped string
*/
function swap($a, $i, $j)
{
    $temp;
    $charArray = str_split($a);
    $temp = $charArray[$i] ;
    $charArray[$i] = $charArray[$j];
    $charArray[$j] = $temp;
    return implode($charArray);
}

// Driver Code
$str = "ABC";
$n = strlen($str);
permute($str, 0, $n - 1);

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

```
?>
```

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 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.
Permutations of a given string using STL

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/