# CS 351 – Design & Analysis of Algorithms Lab

**1)**

**AIM:** Find the min-max of list of elements using divide and conquer strategy.

**Source code:**

def maxmin(a, x, y):

maxi, mini = 0, 0

d = e = f = [0, 0]

if y-x <= 1:

f[0] = max(a[int(x)], a[int(y)])

f[1] = min(a[int(x)], a[int(y)])

else:

d = maxmin(a, x, (x + y)/2)

e = maxmin(a, ((x + y)/2)+1, y)

d[0] = max(d[0], e[0])

d[1] = min(d[1], e[1])

return d

x = int(input('enter number of elements: '))

y = [eval(x) for x in input('enter elements ').split()]

o = maxmin(y, 0, len(y)-1)

print('The maximum and minimum elements in given numbers are: ', \*o)

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input:**

enter number of elements: 10

enter elements 34678 42386 8274 2487625 27867 247864 872464 7864 478468 24786784

**Actual Output:**

The maximum and minimum elements in given numbers are: 24786784 7864

**Expected Output:**

The maximum and minimum elements in given numbers are: 24786784 7864

**TEST CASE 2:**

**Input:**

enter number of elements: 25

enter elements 34 3476 43 32 243 6356 67 5235 5546 55346 3423 356 43567 34767 3467865 45788876 456643 34765 346 34657 46545 43534 34565 345433 7876566

**Actual Output:**

The maximum and minimum elements in given numbers are: 45788876 32

**Expected Output:**

The maximum and minimum elements in given numbers are: 45788876 32

**TEST CASE 3:**

**Input:**

enter number of elements: 6

enter elements 2 2 2 2 2 2

**Actual Output:**

The maximum and minimum elements in given numbers are: 2 2

**Excepted Output:**

The maximum and minimum elements in given numbers are: 2 2

**2)**

**AIM:** Find the kth smallest element using divide and conquer approach.

**Souce code:**

def merge(l):

if len(l) > 1 :

mid=len(l)//2

L=l[:mid]

R=l[mid:]

merge(L)

merge(R)

i=j=k=0

while ((i < len(L)) and (j < len(R))):

if (L[i] < R[j]) :

l[k] = L[i]

i += 1

k += 1

elif (L[i] > R[j]) :

l[k] = R[j]

j += 1

k += 1

else:

l[k] = L[i]

i += 1

j += 1

k += 1

while(i < len(L)):

l[k] = L[i]

i += 1

k += 1

while(j < len(R)):

l[k] = R[j]

j += 1

k += 1

def display(l):

for i in l:

print(i,end=" ")

print()

x = int(input('enter number of elements: '))

y = [eval(x) for x in input('enter elements: ').split()]

k = int(input('enter k: '))

merge(y)

print(f'The {k} smallest element is {y[k-1]}')

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input:**

enter number of elements: 45

enter elements: 14392324 5299112 5315720 19515761 21342955 6499753 11756843 19608503 9310903 19108703 18274971 19306794 48080 5530658 21734970 7493205 5615354 20290911 14931169 20628179 13705914 18582296 5678876 5101209 2735042 20708081 5247136 20092165 4735390 1572990 22740163 6949090 11871876 17273208 4813946 11752948 12846749 11916345 13403663 13870090 6629573 11834638 11367885 3918735 21312731

enter k: 7

**Actual Output:**

The 7 smallest element is 5101209

**Expected Output:**

The 7 smallest element is 5101209

**TEST CASE 2:**

**Input:**

enter number of elements: 100

enter elements: 16206799 2184537 904774 12313842 21895256 22532628 10929479 19330992 20827995 5512010 15752168 5615517 6290570 4696800 20738246 15057828 5801227 5871355 4673580 8276161 10708687 17896363 19992312 1813434 21428546 1477233 21432560 12079432 4802120 458808 5351632 9613873 9328977 4622389 1756112 22346436 18536823 10436656 4274292 12326807 3434244 13455782 13392929 9605831 22270805 10962650 3595852 1770514 12802279 318601 11813438 4112607 2750536 4237683 15222115 10128071 9028397 16207562 14732160 22213729 18532862 1286525 11341911 17323415 12404442 6792262 3264864 12577309 4822035 22373298 9722365 1018470 13858058 19711298 19362469 4089097 20608879 7270655 8606771 23294432 18617910 2849039 22713442 16989237 9602874 22406765 22205148 10391914 7549646 14262237 934137 12599469 10863779 2454368 13217828 16676652 17422715 5182429 3405257 13577316

enter k: 22

**Actual Output:**

The 22 smallest element is 4274292

**Expected Output:**

The 22 smallest element is 4274292

**3)**

**AIM:** Determine the path length from a source vertex to the other vertices in a given graph. (Dijkstra’s algorithm)

**Source Code:**

def Display( dist):

print("\nvertex \tsource distance")

[print(f'{i}\t{dist[i]}') for i in range(v)]

def minDistance( dist, sptSet):

min = float('inf')

for i in range(v):

if dist[i] < min and sptSet[i] == False:

min = dist[i]

mindex = i

return mindex

def dijkstra( src):

dist = [float('inf')] \* v

dist[src] = 0

sptSet = [False] \* v

for \_ in range(v):

x = minDistance(dist, sptSet)

sptSet[x] = True

for y in range(v):

if graph[x][y] > 0 and sptSet[y] == False and dist[y] > dist[x] + graph[x][y]:

dist[y] = dist[x] + graph[x][y]

Display(dist)

v=int(input('enter number of vertices: '))

graph=[([eval(x) for x in input().split()]) for i in range(v)]

dijkstra(int(input('give source: ')))

**RESULT:**

The program runs successfully.

**TEST CASE 1:**

**Input:**

enter number of vertices: 9

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

give source: 2

**Actual Output:**

vertex source distance

0 12

1 8

2 0

3 7

4 14

5 4

6 6

7 7

8 2

**Expected Output:**

vertex source distance

0 12

1 8

2 0

3 7

4 14

5 4

6 6

7 7

8 2

**TEST CASE 2:**

**Input:**

enter number of vertices: 9

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

give source: 0

**Actual Output:**

vertex source distance

0 0

1 4

2 12

3 19

4 21

5 11

6 9

7 8

8 14

**Expected Output:**

vertex source distance

0 0

1 4

2 12

3 19

4 21

5 11

6 9

7 8

8 14

**4)**

**AIM:** Find Shortest path from any node to any other node (All pairs shortest path) within a graph

**Source code:**

def paths(path, v, u, route):

if path[v][u] == v: return

paths(path, v, path[v][u], route)

route.append(path[v][u])

def display(path, n):

for v in range(n):

for u in range(n):

if u != v and path[v][u] != -1:

route = [v]

paths(path, v, u, route)

route.append(u)

print(f'shortest path from {v} to {u}: ', \*route)

def floyd(G,n):

if not G: return

cost = G.copy()

path = [[None for x in range(n)] for y in range(n)]

for v in range(n):

for u in range(n):

if v == u: path[v][u] = 0

elif cost[v][u] != float('inf'): path[v][u] = v

else: path[v][u] = -1

for k in range(n):

for v in range(n):

for u in range(n):

if cost[v][k] != float('inf') and cost[k][u] != float('inf') and (cost[v][k] + cost[k][u] < cost[v][u]):

cost[v][u] = cost[v][k] + cost[k][u]

path[v][u] = path[k][u]

display(path, n)

ver = int(input('enter number of vertices of graph: '))

inf = float('inf')

G=[[eval(x) for x in input().split()] for i in range(ver)]

for i in range(ver):

for j in range(ver):

if G[i][j]==0 and i!=j: G[i][j]=inf

floyd(G,ver)

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input:**

enter number of vertices of graph: 9

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

**Actual Ouput:**

shortest path from 0 to 1: 0 1

shortest path from 0 to 2: 0 1 2

shortest path from 0 to 3: 0 1 2 3

shortest path from 0 to 4: 0 7 6 5 4

shortest path from 0 to 5: 0 7 6 5

shortest path from 0 to 6: 0 7 6

shortest path from 0 to 7: 0 7

shortest path from 0 to 8: 0 1 2 8

shortest path from 1 to 0: 1 0

shortest path from 1 to 2: 1 2

shortest path from 1 to 3: 1 2 3

shortest path from 1 to 4: 1 2 5 4

shortest path from 1 to 5: 1 2 5

shortest path from 1 to 6: 1 7 6

shortest path from 1 to 7: 1 7

shortest path from 1 to 8: 1 2 8

shortest path from 2 to 0: 2 1 0

shortest path from 2 to 1: 2 1

shortest path from 2 to 3: 2 3

shortest path from 2 to 4: 2 5 4

shortest path from 2 to 5: 2 5

shortest path from 2 to 6: 2 5 6

shortest path from 2 to 7: 2 5 6 7

shortest path from 2 to 8: 2 8

shortest path from 3 to 0: 3 2 1 0

shortest path from 3 to 1: 3 2 1

shortest path from 3 to 2: 3 2

shortest path from 3 to 4: 3 4

shortest path from 3 to 5: 3 2 5

shortest path from 3 to 6: 3 2 5 6

shortest path from 3 to 7: 3 2 5 6 7

shortest path from 3 to 8: 3 2 8

shortest path from 4 to 0: 4 5 6 7 0

shortest path from 4 to 1: 4 5 2 1

shortest path from 4 to 2: 4 5 2

shortest path from 4 to 3: 4 3

shortest path from 4 to 5: 4 5

shortest path from 4 to 6: 4 5 6

shortest path from 4 to 7: 4 5 6 7

shortest path from 4 to 8: 4 5 2 8

shortest path from 5 to 0: 5 6 7 0

shortest path from 5 to 1: 5 2 1

shortest path from 5 to 2: 5 2

shortest path from 5 to 3: 5 2 3

shortest path from 5 to 4: 5 4

shortest path from 5 to 6: 5 6

shortest path from 5 to 7: 5 6 7

shortest path from 5 to 8: 5 2 8

shortest path from 6 to 0: 6 7 0

shortest path from 6 to 1: 6 7 1

shortest path from 6 to 2: 6 5 2

shortest path from 6 to 3: 6 5 2 3

shortest path from 6 to 4: 6 5 4

shortest path from 6 to 5: 6 5

shortest path from 6 to 7: 6 7

shortest path from 6 to 8: 6 8

shortest path from 7 to 0: 7 0

shortest path from 7 to 1: 7 1

shortest path from 7 to 2: 7 6 5 2

shortest path from 7 to 3: 7 6 5 2 3

shortest path from 7 to 4: 7 6 5 4

shortest path from 7 to 5: 7 6 5

shortest path from 7 to 6: 7 6

shortest path from 7 to 8: 7 8

shortest path from 8 to 0: 8 2 1 0

shortest path from 8 to 1: 8 2 1

shortest path from 8 to 2: 8 2

shortest path from 8 to 3: 8 2 3

shortest path from 8 to 4: 8 2 5 4

shortest path from 8 to 5: 8 2 5

shortest path from 8 to 6: 8 6

shortest path from 8 to 7: 8 7

**Expected Output:**

shortest path from 0 to 1: 0 1

shortest path from 0 to 2: 0 1 2

shortest path from 0 to 3: 0 1 2 3

shortest path from 0 to 4: 0 7 6 5 4

shortest path from 0 to 5: 0 7 6 5

shortest path from 0 to 6: 0 7 6

shortest path from 0 to 7: 0 7

shortest path from 0 to 8: 0 1 2 8

shortest path from 1 to 0: 1 0

shortest path from 1 to 2: 1 2

shortest path from 1 to 3: 1 2 3

shortest path from 1 to 4: 1 2 5 4

shortest path from 1 to 5: 1 2 5

shortest path from 1 to 6: 1 7 6

shortest path from 1 to 7: 1 7

shortest path from 1 to 8: 1 2 8

shortest path from 2 to 0: 2 1 0

shortest path from 2 to 1: 2 1

shortest path from 2 to 3: 2 3

shortest path from 2 to 4: 2 5 4

shortest path from 2 to 5: 2 5

shortest path from 2 to 6: 2 5 6

shortest path from 2 to 7: 2 5 6 7

shortest path from 2 to 8: 2 8

shortest path from 3 to 0: 3 2 1 0

shortest path from 3 to 1: 3 2 1

shortest path from 3 to 2: 3 2

shortest path from 3 to 4: 3 4

shortest path from 3 to 5: 3 2 5

shortest path from 3 to 6: 3 2 5 6

shortest path from 3 to 7: 3 2 5 6 7

shortest path from 3 to 8: 3 2 8

shortest path from 4 to 0: 4 5 6 7 0

shortest path from 4 to 1: 4 5 2 1

shortest path from 4 to 2: 4 5 2

shortest path from 4 to 3: 4 3

shortest path from 4 to 5: 4 5

shortest path from 4 to 6: 4 5 6

shortest path from 4 to 7: 4 5 6 7

shortest path from 4 to 8: 4 5 2 8

shortest path from 5 to 0: 5 6 7 0

shortest path from 5 to 1: 5 2 1

shortest path from 5 to 2: 5 2

shortest path from 5 to 3: 5 2 3

shortest path from 5 to 4: 5 4

shortest path from 5 to 6: 5 6

shortest path from 5 to 7: 5 6 7

shortest path from 5 to 8: 5 2 8

shortest path from 6 to 0: 6 7 0

shortest path from 6 to 1: 6 7 1

shortest path from 6 to 2: 6 5 2

shortest path from 6 to 3: 6 5 2 3

shortest path from 6 to 4: 6 5 4

shortest path from 6 to 5: 6 5

shortest path from 6 to 7: 6 7

shortest path from 6 to 8: 6 8

shortest path from 7 to 0: 7 0

shortest path from 7 to 1: 7 1

shortest path from 7 to 2: 7 6 5 2

shortest path from 7 to 3: 7 6 5 2 3

shortest path from 7 to 4: 7 6 5 4

shortest path from 7 to 5: 7 6 5

shortest path from 7 to 6: 7 6

shortest path from 7 to 8: 7 8

shortest path from 8 to 0: 8 2 1 0

shortest path from 8 to 1: 8 2 1

shortest path from 8 to 2: 8 2

shortest path from 8 to 3: 8 2 3

shortest path from 8 to 4: 8 2 5 4

shortest path from 8 to 5: 8 2 5

shortest path from 8 to 6: 8 6

shortest path from 8 to 7: 8 7

**5)**

**AIM:** Find the non attacking positions of Queens in a given chess board using backtracking Technique.

**Source code:**

def isSafe(mat, r, c):

for i in range(r):

if mat[i][c] == 1: return False

i, j = r, c

while i >= 0 and j >= 0:

if mat[i][j] == 1: return False

i -= 1

j -= 1

i, j = r, c

while i >= 0 and j < len(mat):

if mat[i][j] == 1: return False

i -= 1

j += 1

return True

def nQueen(mat, r):

if r == len(mat):

for r in mat: print(\*r)

print()

return

for i in range(len(mat)):

if isSafe(mat, r, i):

mat[r][i] = 1

nQueen(mat, r + 1)

mat[r][i] = 0

n = int(input('enter number of queens: '))

if n<4: print('there is no such arrangement')

else:

mat = [[0 for \_ in range(n)] for \_ in range(n)]

nQueen(mat, 0)

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input:**

enter number of queens: 4

**Acutal Ouput:**

0 1 0 0

0 0 0 1

1 0 0 0

0 0 1 0

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

**Expected Output:**

0 1 0 0

0 0 0 1

1 0 0 0

0 0 1 0

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

**TEST CASE 2:**

**Input:**

enter number of queens: 2

there is no such arrangement

**Acutal Ouput:**

there is no such arrangement

**Expected Output:**

there is no such arrangement

**6)**

**AIM:** Calculate the optimal profit of a Knapsack using Branch and Bound Technique.

**Source code:**

n=int(input('enter number of items: '))

val=[-1\*eval(x) for x in input('enter values/profits of items: ').split()]

wt=[eval(x) for x in input('enter weights of items: ').split()]

W=int(input('enter knapsack capacity: '+'\n'))

knap=[1]\*n

w=lb=0

for j in range(n):

if w+wt[j]<=W:

w+=wt[j]

lb+=val[j]

upper=lb

lb+=((val[j]/wt[j])\*(W-w))

for i in range(n):

lb1=up1=0

w1=0

for j in range(n):

if i!=j and w1+wt[j]<=W and knap[j]==1:

w1+=wt[j]

lb1+=val[j]

up1=lb1

lb1+=((val[j]/wt[j])\*(W-w1))

if upper>lb1:

if upper>up1:

knap[i]=0

upper=up1

elif lb1<lb:

knap[i]=0

lb=lb1

elif lb1==lb:

if upper>up1:

knap[i]=0

upper=up1

lb=lb1

print(-1\*upper)

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input:**

enter number of items: 4

enter values/profits of items: 10 10 12 18

enter weights of items: 2 4 6 9

enter knapsack capacity: 10

**Actual Ouput:**

22

**Expected Output:**

22

**TEST CASE 2:**

**Input:**

enter number of items: 3

enter values/profits of items: 60 100 120

enter weights of items: 10 20 30

enter knapsack capacity: 50

**Actual Ouput:**

220

**Expected Output:**

220

## HACKER RANK PROBLEMS

**1)**

**AIM:** A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. Longest common subsequence (LCS) of 2 sequences is a subsequence, with maximal length, which is common to both the sequences.  
  
Given two sequences of integers, A=[a[1],a[2],a[3],……,a[n]] and B=[b[1],b[2],b[3],……,b[n]] find the longest common subsequence and print it as a line of space-separated integers. If there are multiple common subsequences with the same maximum length, print any one of them.

In case multiple solutions exist, print any of them. It is guaranteed that at least one non-empty common subsequence will exist.

**Source code:**

import math

import os

import random

import re

import sys

def longestCommonSubsequence(a, b):

# Write your code heree

n, m = len(a), len(b)

dp = [[0 for i in range(m + 1)] for j in range(n + 1)]

'''for i in range(n + 1):

for j in range(m + 1):

if i == 0 or j == 0:

dp[i][j] = 0'''

for i in range(1, n + 1):

for j in range(1, m + 1):

if a[i - 1] == b[j - 1]:

dp[i][j] = 1 + dp[i - 1][j - 1]

else:

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

i, j, v = n, m, []

while i > 0 and j > 0:

if a[i - 1] == b[j - 1]:

v.append(a[i - 1])

i -= 1

j -= 1

else:

if dp[i - 1][j] > dp[i][j - 1]:

i -= 1

else :

j -= 1

return reversed(v)

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

first\_multiple\_input = input().rstrip().split()

n = int(first\_multiple\_input[0])

m = int(first\_multiple\_input[1])

a = list(map(int, input().rstrip().split()))

b = list(map(int, input().rstrip().split()))

result = longestCommonSubsequence(a, b)

fptr.write(' '.join(map(str, result)))

fptr.write('\n')

fptr.close()

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input (stdin)**

5 6

1 2 3 4 1

3 4 1 2 1 3

**Actual Output (stdout)**

3 4 1

**Expected Output**

3 4 1

**TEST CASE 2:**

**Input (stdin):**

9 10

3 9 8 3 9 7 9 7 0

3 3 9 9 9 1 7 2 0 6

**Actual Output (stdout):**

3 3 9 9 7 0

**Expected Output:**

3 3 9 9 7 0

**2)**

**AIM:** Given an amount and the denominations of coins available, determine how many ways change can be made for amount. There is a limitless supply of each coin type.

**Source code:**

import math

import os

import random

import re

import sys

def getWays(n, c):

clist=[1]+[0]\*n

for coin in c:

for i in range(coin,n+1):

clist[i]+=clist[i-coin]

return clist[n]

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

first\_multiple\_input = input().rstrip().split()

n = int(first\_multiple\_input[0])

m = int(first\_multiple\_input[1])

c = list(map(int, input().rstrip().split()))

ways = getWays(n, c)

fptr.write(str(ways) + '\n')

fptr.close()

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input (stdin):**

4 3

1 2 3

**Actual Output (stdout):**

4

**Expected Output:**

4

**TEST CASE 2:**

**Input (stdin):**

10 4

2 5 3 6

**Actual Output (stdout):**

5

**Expected Output:**

5

**3)**

**AIM:** A group of friends want to buy a bouquet of flowers. The florist wants to maximize his number of new customers and the money he makes. To do this, he decides he'll multiply the price of each flower by the number of that customer's previously purchased flowers plus 1. The first flower will be original price, (0+1) x original price , the next will be, (1+1) x original price   and so on.

Given the size of the group of friends, the number of flowers they want to purchase and the original prices of the flowers, determine the minimum cost to purchase all of the flowers. The number of flowers they want equals the length of the c array.

**Source code:**

import math

import os

import random

import re

import sys

def getMinimumCost(k, c):

sumi=0

n=len(c)

if(k==n):

return(sum(c))

else:

j=0

while(c!=[]):

for i in range(k):

if c==[]:

break

else:

sumi+=(j+1)\*max(c)

c.remove(max(c))

j+=1

return(sumi)

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

nk = input().split()

n = int(nk[0])

k = int(nk[1])

c = list(map(int, input().rstrip().split()))

minimumCost = getMinimumCost(k, c)

fptr.write(str(minimumCost) + '\n')

fptr.close()

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input (stdin):**

3 3

2 5 6

**Actual Output (stdout):**

13

**Expected Output:**

13

**TEST CASE 2:**

**Input (stdin):**

3 2

2 5 6

**Actual Output (stdout):**

15

**Expected Output:**

15

**TEST CASE 3:**

**Input (stdin):**

5 3

1 3 5 7 9

**Actual Output (stdout):**

29

**Expected Output:**

29

**4)**

**AIM:** You will be given a list of integers, arr, and a single integer k . You must create an array of length k from elements of arr such that its unfairness is minimized. Call that array’. Unfairness of an array is calculated as *max( arr ’) – min( arr’ )*

Where:  
- max denotes the largest integer in arr’.  
- min denotes the smallest integer in arr’.

**Source code:**

import math

import os

import random

import re

import sys

import itertools

def maxMin(k, arr):

arr.sort()

return min((arr[i-1]-arr[i-k]) for i in range(k,len(arr)+1))

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

n = int(input().strip())

k = int(input().strip())

arr = []

for \_ in range(n):

arr\_item = int(input().strip())

arr.append(arr\_item)

result = maxMin(k, arr)

fptr.write(str(result) + '\n')

fptr.close()

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input (stdin):**

7

3

10

100

300

200

1000

20

30

**Actual Output (stdout):**

20

**Expected Output:**

20

**TEST CASE 2:**

**Input (stdin):**

10

4

1

2

3

4

10

20

30

40

100

200

**Actual Output (stdout):**

3

**Expected Output:**

3

**TEST CASE 3:**

**Input (stdin):**

5

2

1

2

1

2

1

**Actual Output (stdout):**

0

**Expected Output:**

0

**5)**

**AIM:** Consider an undirected graph where each edge weighs 6 units. Each of the nodes is labeled consecutively from 1 to n.

You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the breadth-first search algorithm ([BFS](https://en.wikipedia.org/wiki/Breadth-first_search)). Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.

**Source code:**

import math

import os

import random

import re

import sys

from collections import deque

from collections import defaultdict

def bfs(n, m, edges, s):

# Write your code here

graph=defaultdict(set)

for l, r in edges:

graph[l].add(r)

graph[r].add(l)

visited = {}

# Explore graph once

frontier = deque([(s, 0)])

seen = {s}

while frontier:

curr\_node, curr\_cost = frontier.popleft()

for nbour in graph[curr\_node]:

if nbour not in seen:

seen.add(nbour)

visited[nbour] = curr\_cost+6

frontier.append((nbour, curr\_cost+6))

result = []

for node in range(1, n+1):

if s != node:

result.append(visited.get(node, -1))

return result

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

q = int(input().strip())

for q\_itr in range(q):

first\_multiple\_input = input().rstrip().split()

n = int(first\_multiple\_input[0])

m = int(first\_multiple\_input[1])

edges = []

for \_ in range(m):

edges.append(list(map(int, input().rstrip().split())))

s = int(input().strip())

result = bfs(n, m, edges, s)

fptr.write(' '.join(map(str, result)))

fptr.write('\n')

fptr.close()

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input (stdin);**

2

4 2

1 2

1 3

1

3 1

2 3

2

**Actual Output (stdout):**

6 6 -1

-1 6

**Expected Output:**

6 6 -1

-1 6

**TEST CASE 2:**

**Input (stdin):**

1

5 3

1 2

1 3

3 4

1

**Actual Output (stdout):**

6 6 12 -1

**Expected Output:**

6 6 12 -1

**6)**

**AIM:** Given an undirected weighted connected graph, find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and:

* There is only one exclusive path from a node to every other node.
* The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
* No cycles are formed

To create the Really Special SubTree, always pick the edge with smallest weight. Determine if including it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

* Choose the edge that minimizes the sum  where  and  are vertices and  is the edge weight.
* If there is still a collision, choose any of them.

Print the overall weight of the tree formed using the rules.

**Source code:**

import sys

def root(vertex):

global rootsIDs

if rootsIDs[vertex]==vertex:

return vertex

else:

while(rootsIDs[vertex]!=vertex):

rootsIDs[vertex]=rootsIDs[rootsIDs[vertex]]

vertex=rootsIDs[vertex]

return vertex

def union(x,y):

global rootsIDs

Xroot=root(x)

Yroot=root(y)

rootsIDs[Yroot]=rootsIDs[Xroot]

return

def Kruskal(GE):

global rootsIDs

minimumWeight=0

for edge in GE:

x=edge[0]

y=edge[1]

weight=edge[2]

if root(x)!=root(y):

minimumWeight+=weight

union(x,y)

return minimumWeight

if \_\_name\_\_ == '\_\_main\_\_':

V, E = map(int, input().split())

#create and fill list of GraphEges

GE=[]

for i in range(E):

GE.append(list(map(int, input().split())))

GE=sorted(GE, key=lambda x:x[2])

rootsIDs=list(range(V+1))

print(Kruskal(GE))

**RESULT:**

The program executes successfully.

**TEST CASE 1:**

**Input (stdin):**

4 6

1 2 5

1 3 3

4 1 6

2 4 7

3 2 4

3 4 5

**Actual Output (stdout):**

12

**Expected Output:**

12

**TEST CASE 2:**

**Input (stdin):**

5 7

1 2 20

1 3 50

1 4 70

1 5 90

2 3 30

3 4 40

4 5 60

**Actual Output (stdout):**

150

**Expected Output:**

150