Department of CSE, BMSCE
ADA Lab program list for Test-2
Aug-Dec 2019

1. Using Dynamic Programming technique, write program for Printing Longest Common Subsequence (LCS): Given two sequences, print the longest subsequence present in both of them.
Examples:
LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.
LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

```
#include<iostream>
#include<string>
#include<algorithm>

using namespace std;

int main()
{
	string x,y;
	int m,n,count;
	cout<<"Enter string1:";
	getline(cin,x);
	cout<<"Enter string2:";
	getline(cin,y);
	m=x.length();
	n=y.length();

	int a[m+1][n+1]={-1};
	for(int i=0;i<=m;i++)
	{
		for(int j=0;j<=n;j++)
		{
			if((i==0)||(j==0))
				a[i][j]=0;
			else if(x[i-1]==y[j-1])
			{	a[i][j]=a[i-1][j-1]+1;}
			else
				a[i][j]=max(a[i-1][j],a[i][j-1]);
		}
	}
	int index = a[m][n];

char lcs[index+1];

lcs[index] = '\0'; // Set the terminating character

//Start from the right-most-bottom-most corner and one by one store characters in lcs[]

int i = m, j = n;

while (i > 0 && j > 0)

{

if (x[i-1] == y[j-1])
```

```
        {
        lcs[index-1] = x[i-1]; // Put current character in result
        i--; j--; index--;
        }
        else if (a[i-1][j] > a[i][j-1])
        i--;
        else
        j--;
        }
            cout<<endl<<"length:"<<a[m][n]<<endl<<lcs;
            return 0;
}
```

2. Using Dynamic Programming technique, write program to solve 0-1 knapsack problem. Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item, or don't pick it (0-1 property).
Example:

```
value[] = {60, 100, 120};
weight[] = {10, 20, 30};
W = 50;

Solution: 220
```

```
#include<iostream>
#include<algorithm>

using namespace std;

int n;

int Knapsack(int w[],int v[],int C)
{
   int i,j,V[50][50];
   for(int i=0;i<=n;i++)
```

```
        V[i][0]=0;

    for(int j=0;i<=C;i++)
        V[0][j]=0;

    for(int i=1;i<=n;i++)
        for(int j=1;j<=C;j++)
        {
            if((j-w[i])>=0)
                V[i][j]=(V[i-1][j]>(v[i]+V[i-1][j-w[i]]))?V[i-1][j]:(v[i]+V[i-1][j-w[i]]);
            else
                V[i][j]=V[i-1][j];
        }
    for(int i=0;i<=n;i++){
        for(int j=0;j<=C;j++)
            cout<<V[i][j]<<" ";
        cout<<endl;
    }
    return V[n][C];
}

int main()
{
    int C;
    cout<<"Enter the number of items: ";
    cin>>n;
    cout<<"Enter the capacity: ";
    cin>>C;
    int *w=new int[n+1];
    int *v=new int[n+1];
    w[0]=0;
    v[0]=0;
    cout<<"Enter the weight of the items:"<<endl;
    for(int i=1;i<=n;i++)
        cin>>w[i];

    cout<<"Enter the value of the items:"<<endl;
    for(int i=1;i<=n;i++)
        cin>>v[i];

    int value=Knapsack(w,v,C);
    cout<<"Highest value possible: "<<value;

    return 0;
}
```

3. Using Dynamic Programming technique, write program to find minimum Edit distance. Given two text strings A of length n and B of length m, to transform A into B using a minimum number of operations of the following types: **delete** a character from A, **insert** a character into A, or **change** some character in A into a new character. The minimal number of such operations required to transform A into B is called the edit distance between A and B. For example, the edit distance(abbc, xbbacd) = 3 (change a to x; insert a after bb; insert d after c) Find a recurrence relation that is useful to find the edit distance of two strings A and B.
Examples:
Input: str1 = "geek", str2 = "gesek"
Output: 1
We can convert str1 into str2 by inserting a 's'.

Input: str1 = "cat", str2 = "cut"
Output: 1
We can convert str1 into str2 by replacing 'a' with 'u'.

Input: str1 = "sunday", str2 = "saturday"
Output: 3
Last three and first characters are same. We basically need to convert "un" to "atur". This can be done using below three operations. Replace 'n' with 'r', insert t, insert a

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


int min(int x, int y, int z)
{
  return min(min(x, y), z);
}

int editDist(string str1 , string str2 , int m ,int n)
{

   if (m == 0) return n;

   if (n == 0) return m;
   if (str1[m-1] == str2[n-1])
      return editDist(str1, str2, m-1, n-1);

   return 1 + min ( editDist(str1,  str2, m, n-1),    // Insert
                    editDist(str1,  str2, m-1, n),   // Remove
                    editDist(str1,  str2, m-1, n-1) // Replace
                  );
}


int main()
{
   // your code goes here
   string str1,str2;
   cout<<"Enter string1 and Strin2 values";
   cin>>str1>>str2;

   cout << editDist( str1 , str2 , str1.length(), str2.length());
```

```
   return 0;
}
```

4. Using the concept of Transform and Conquer technique, Write program to find smallest range containing elements from k lists. Given k sorted lists of integers of size n each, find the smallest range that includes at least element from each of the k lists. If more than one smallest ranges are found, print any one of them.

Example:

Input:

K = 3
arr1[] : [4, 7, 9, 12, 15]
arr2[] : [0, 8, 10, 14, 20]
arr3[] : [6, 12, 16, 30, 50]

Output:

The smallest range is [6 8]

Input:

k = 3
arr1[] : [4, 7]
arr2[] : [1, 2]
arr3[] : [20, 40]

Output

The smallest range is [2 20]

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

using namespace std;
int ptr[501];
void findSmallestRange(int **arr, int n, int k)
{
        int i,minval,maxval,minrange,minel,maxel,flag,minind;

        for(i = 0;i <= k;i++)
                ptr[i] = 0;

        minrange = INT_MAX;

        while(1)
        {
                minind = -1;
                minval = INT_MAX;
                maxval =        INT_MIN;
                flag = 0;


                for(i = 0;i < k;i++)
                {

                        if(ptr[i] == n)
                        {
                                flag = 1;
```

```
                        break;
                }

                if(ptr[i] < n && arr[i][ptr[i]] < minval)
                {
                        minind=i;
                        minval=arr[i][ptr[i]];
                }

                if(ptr[i] < n && arr[i][ptr[i]] > maxval)
                {
                        maxval = arr[i][ptr[i]];
                }
        }

        //Program exited with return code: 0

        if(flag)
                break;

        ptr[minind]++;


        if((maxval-minval) < minrange)
        {
                minel = minval;
                maxel = maxval;
                minrange = maxel - minel;
        }
    }

    printf("The smallest range is [%d , %d]\n",minel,maxel);
}

int main()
{
    int k,N;
    cout<<"ENTER THE VALUE OF K:\n ";
    cin>>k;
    cout<<"\nENTER THE SIZE OF THE ARRAY:\n";
    cin>>N;
    int **arr=new int*[k];
    for(int i=0;i<k;i++)
        arr[i]=new int[N];
    cout<<"\nENTER THE ARRAY ELEMENTS:\n";
    for(int i=0;i<k;i++)
        for(int j=0;j<N;j++)
            cin>>arr[i][j];

    findSmallestRange(arr,N,k);

    return 0;
}
```

5. Using Dynamic Programming technique, write Program to solve Minimum Coin Changing Problem. In this problem the goal is to make change for an amount using least number of coins from the available denominations.

Example

Say I went to a shop and bought 4 toffees. It cost me Rs. 4 in total. So, I gave Rs. 10 to the shopkeeper.

The shopkeeper had Rs. 1 coins, Rs. 2 coins and Rs. 5 coins with him. Now, the goal is: The shopkeeper has to make change for Rs. 6 using least number of coins from the available denominations coins (1, 2 and 5)

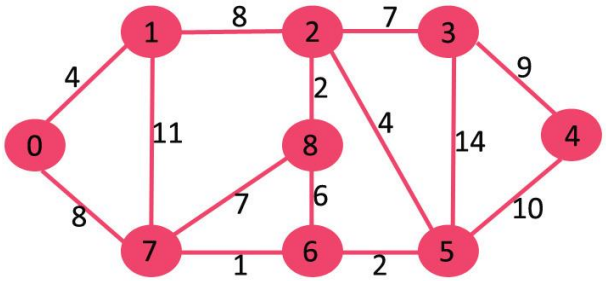
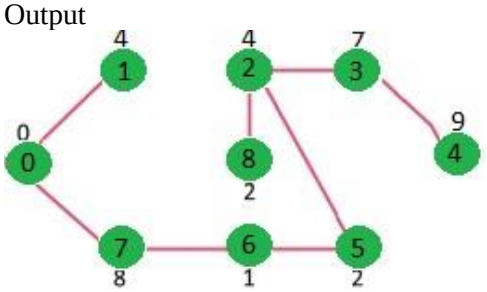
Answer

Therefore, to make change of amount Rs. 6 the shopkeeper will need minimum 2 coins and the coins will be Rs. 1 and Rs. 5

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

int coin_index[10];
int minCoins(int coins[], int m, int V)
{

        int table[V+1];


        table[0] = 0;
   coin_index[0]=0;

        for (int i=1; i<=V; i++)
                table[i] = INT_MAX;

        for (int i=1; i<=V; i++)
        {
                for (int j=0; j<m; j++)
                if (coins[j] <= i)
                {
                        int sub_res = table[i-coins[j]];
                        if (sub_res != INT_MAX && sub_res + 1 < table[i])
                        {
                                table[i] = sub_res + 1;
                                coin_index[i] = j;
         }
                }
        }
        return table[V];
}


int main()
{
        int coins[] = {5,2,1};
        int m = sizeof(coins)/sizeof(coins[0]);
        int V;
        cout<<"Enter the value of money: ";
        cin>>V;
        cout << minCoins(coins, m, V)<<endl;
```

```
    int v= V;
    while(v>0)
    {
       if(v>0)
          cout<<coins[coin_index[v]]<<" ";
       v = v-coins[coin_index[v]];
    }
          return 0;
}
```

6. Using Prims Algorithm, write program to find Minimum Cost Spanning Tree,

Input	Output
	


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

#define V 5

int minKey(int key[], bool mstSet[])
{

   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;
}
```

```
void printMST(int parent[], int graph[V][V])
{
    int sum=0;
   //cout<<"Edge \tWeight\n";
   for (int i = 1; i < V; i++)
   {
     sum=sum+graph[i][parent[i]];
     // cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";

    }
     cout<<"Minimum weight of spanning tree :"<<sum<<endl;
}


void primMST(int graph[V][V])
{

   int parent[V];


   int key[V];


   bool mstSet[V];


   for (int i = 0; i < V; i++)
      key[i] = INT_MAX, mstSet[i] = false;

   key[0] = 0;
   parent[0] = -1;
```

```
    for (int count = 0; count < V - 1; count++)
    {

        int u = minKey(key, mstSet);

        mstSet[u] = true;


        for (int v = 0; v < V; v++)

            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }


    printMST(parent, graph);
}


int main()
{

    int graph[V][V];
    cout<<"Enter the Graph matrix"<<endl;
    for(int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            cin>>graph[i][j];
        }
```

```
    }

    primMST(graph);

    return 0;
}
```

7. Using Dynamic Programming technique, write program to solve partition problem. Partition problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is same.

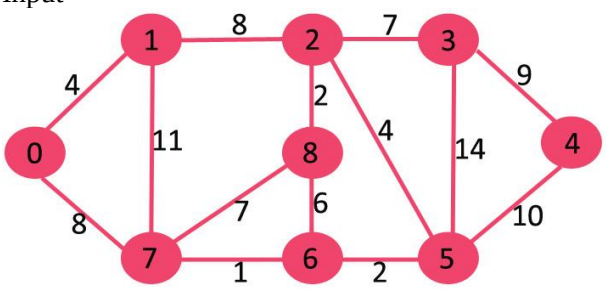
Examples:
arr[] = {1, 5, 11, 5}
Output: true
The array can be partitioned as {1, 5, 5} and {11}
arr[] = {1, 5, 3}
Output: false
The array cannot be partitioned into equal sum sets.

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

bool isSubsetSum (int arr[], int n, int sum)
{

    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;


    if (arr[n-1] > sum)
      return isSubsetSum (arr, n-1, sum);


    return isSubsetSum (arr, n-1, sum) ||
        isSubsetSum (arr, n-1, sum-arr[n-1]);
}


bool findPartiion (int arr[], int n)
{
```

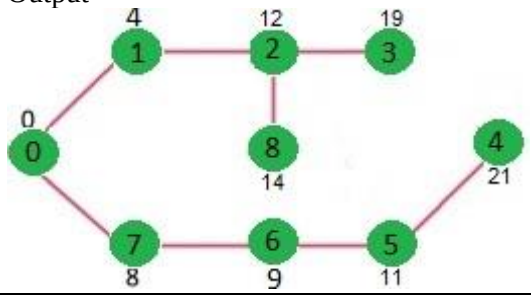
```
    int sum = 0;
    for (int i = 0; i < n; i++)
    sum += arr[i];

    if (sum%2 != 0)
    return false;


    return isSubsetSum (arr, n, sum/2);
}


int main()
{
    int n,i;
    cout<<"Enter the array size";
    cin>>n;
    int arr[n];
    cout<<"Enter the array elements";
    for(i=0;i<n;i++)
    {
        cin>>arr[i];
    }
    if (findPartiion(arr, n) == true)
        cout << "Can be divided into two subsets "
                "of equal sum";
    else
        cout << "Can not be divided into two subsets"
                " of equal sum";
    return 0;
}
```

8. Using Greedy Technique, write program to solve shortest path problem using Dijkstra's algorithm. Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.

Input



Output




```
#include<iostream>

using namespace std;
```

```
int main()
{
	int n,mat[30][30],src,dist[10],vis[10],count = 0,min,prev[10]={0},u;
	cout<<"enter the no of nodes"<<endl;
	cin>>n;
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
		{
			cin>>mat[i][j];
		}
	}
	cout<<"enter the source"<<endl;
	cin>>src;
	for(int j=0;j<n;j++)
		dist[j] = mat[src][j];
	for(int j=0;j<n;j++)
		vis[j] = 0;
	dist[src]=0;
	vis[src] = 1;
	count = 1;

	while(count!=n)
	{
		min = 999;
		for(int i=0;i<n;i++)
			if(dist[i]<min && vis[i]!=1)
			{
				min = dist[i];
				u = i;
			}
		vis[u]=1;
		count++;

		for(int i=0;i<n;i++)
		{
			if(min+mat[u][i]<dist[i] && vis[i]!=1)
			{
				dist[i] = min+mat[u][i];
				prev[i] = u;
			}
		}
	}

	for(int i=0;i<n;i++)
		cout<<"source "<<src<<" vertex "<<i<<" distance "<<dist[i]<<endl;

	for(int i =1;i<n;i++)
	{
		cout<<"Path from source to vertex "<<i<<endl;
		int k =0;
		int j =i;
		while(j!=k)
		{
```

```
                cout<<j<<"<--";
                j = prev[j];
            }
            cout<<0;
            cout<<endl;
        }


}
```

9. Using Dynamic Programming Technique, write program to solve all pairs Shortest Path by Floyd's Algorithm.

Input	Output
$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 3 & 1 & 4 \\ 6 & 0 & 3 & 2 & 5 \\ \mathbf{10} & \mathbf{12} & 0 & 4 & 7 \\ \mathbf{6} & \mathbf{8} & 2 & 0 & 3 \\ 3 & 5 & 6 & 4 & 0 \end{bmatrix}$

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

using namespace std;


#define V 5


#define INF 9999

void printSolution(int dist[][V]);


void floydWarshall (int graph[][V])
```

```
{

    int dist[V][V], i, j, k;



    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];



    for (k = 0; k < V; k++)
    {

        for (i = 0; i < V; i++)
        {

            for (j = 0; j < V; j++)
            {

                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
```

```
    printSolution(dist);
}

void printSolution(int dist[][V])
{
    cout<<"The following matrix shows the shortest distances"
          " between every pair of vertices \n";
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (dist[i][j] == INF)
                cout<<"INF"<<"     ";
            else
                cout<<dist[i][j]<<"     ";
        }
        cout<<endl;
    }
}



int main()
{
            cout<<"Enter The Matrix Values\n";
    int graph[V][V] ;
    int i,j;
```

```
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
        {
            cin>>graph[i][j];
        }
    }

    floydWarshall(graph);
    return 0;
}
```

10. Using Dynamic Programming, write program to solve Longest increasing subsequence. The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.

arr[ ]	10	22	9	33	21	50	41	60	80
LIS	1	2		3		4		5	6

Examples:

Input : arr[] = {3, 10, 2, 1, 20}
Output : Length of LIS = 3
//The longest increasing subsequence is 3, 10, 20

Input : arr[] = {3, 2}
Output : Length of LIS = 1
//The longest increasing subsequences are {3} and {2}

Input : arr[] = {50, 3, 10, 7, 40, 80}
Output : Length of LIS = 4

```
//The longest increasing subsequence is {3, 7, 40, 80}


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


int _lis( int arr[], int n, int *max_ref)
{

   if (n == 1)
      return 1;


   int res, max_ending_here = 1;


   for (int i = 1; i < n; i++)
   {
      res = _lis(arr, i, max_ref);
      if (arr[i-1] < arr[n-1] && res + 1 > max_ending_here)
         max_ending_here = res + 1;
   }


   if (*max_ref < max_ending_here)
     *max_ref = max_ending_here;

   return max_ending_here;
}


int lis(int arr[], int n)
{

   int max = 1;
   _lis( arr, n, &max );

   return max;
}


int main()
{
   int n,i;
   printf("Enter the array size");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the array elements");
   for(i=0;i<n;i++)
   {
      scanf("%d",&arr[i]);
   }
   printf("Length of lis is %dn",
         lis( arr, n ));
```

```
    return 0;
}
```