Sr.No.	**Program Name**	**Date**	**Remarks**
1.	To perform the program of Linear search.		
2.	To perform the program of Binary search.		
3.	To perform the program of sorting of an array using Insertion Sort.		
4.	To perform the program of sorting of an array using Quick sort.		
5.	To perform the program of sorting of an array using Merge sort.		
6 (A).	To perform the program of Fibonacci series using dynamic programming (DP) method.		
6 (B).	To perform the program of Fibonacci series without using dynamic programming (DP) method.		
7.	Write a program for the implementation of Activity selection problem using Greedy Approach.		
8.	Write a program to solve travelling salesman problem.		
9.	C program to implement Longest Common Sub-Sequence.		
10.	Write a program to implement MST of given undirected graph using Prim's algorithm.		
11.	Write a program to implement MST of given undirected graph using Kruskal's algorithm.		
12.	From a given vertex in a weighted connected graph,find shortest paths to other vertices using Dijkistra's algorithm.		
13.	Implement Bellman-Ford Algorithm to find Shortest Path from a source.		
14	Implement All Pairs Shortest Paths Problem Using Floyd's-Warshall Algorithm.		

Program – 06 (B)

AIM: To perform the program of Fibonacci series using dynamic programming (DP) method.

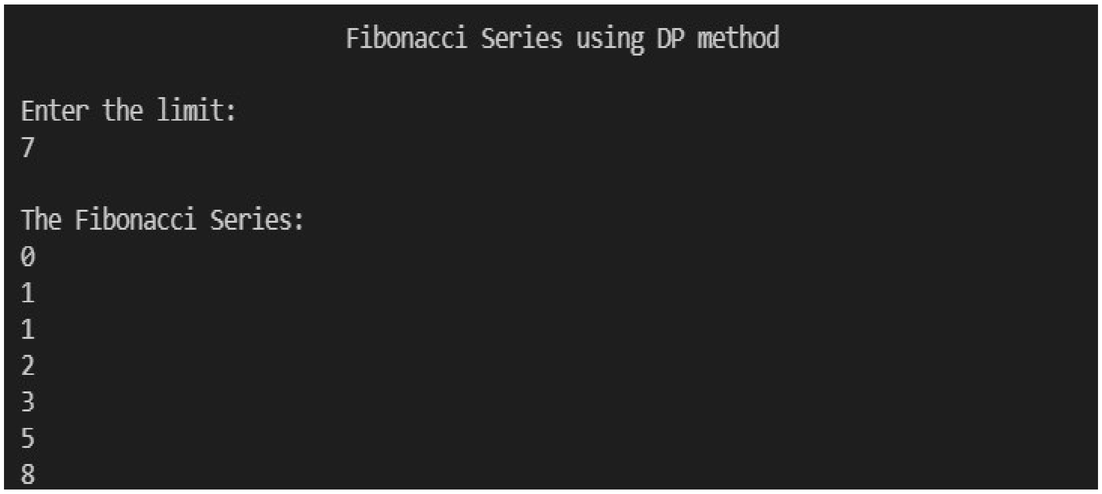
Description: In this series, the next number is the sum of previous two numbers. By using DP method, all the values are calculated once and stored in a table. So, whenever these values are required for calculation, they can be fetched from the table which results in the lesser cost of the program.

Source Code:

```
#include<stdio.h>
#include<conio.h>
int main(int argc, char const *argv[])
{
   int a[10], n, j;
   // clrscr();
   printf("\n\t\t\tFibonacci Series using DP method\n\n");
   printf("Enter the limit:\n");
   scanf("%d", &n);
   a[0] = 0;
   a[1] = 1;
   for(int i = 2; i <=n; i++)
   {
      a[i] = a[i-1]+a[i-2];
   }
   printf("\nThe Fibonacci Series:\n");
   for(int i = 0; i < n; i++)
   {
      printf("%d\n", a[i]);
```

```
    }
    return 0;
}
```

OUTPUT:



```
                         Fibonacci Series using DP method

Enter the limit:
7

The Fibonacci Series:
0
1
1
2
3
5
8
```

Program – 06 (A)

AIM: To perform the program of Fibonacci series without using dynamic programming(DP) method.

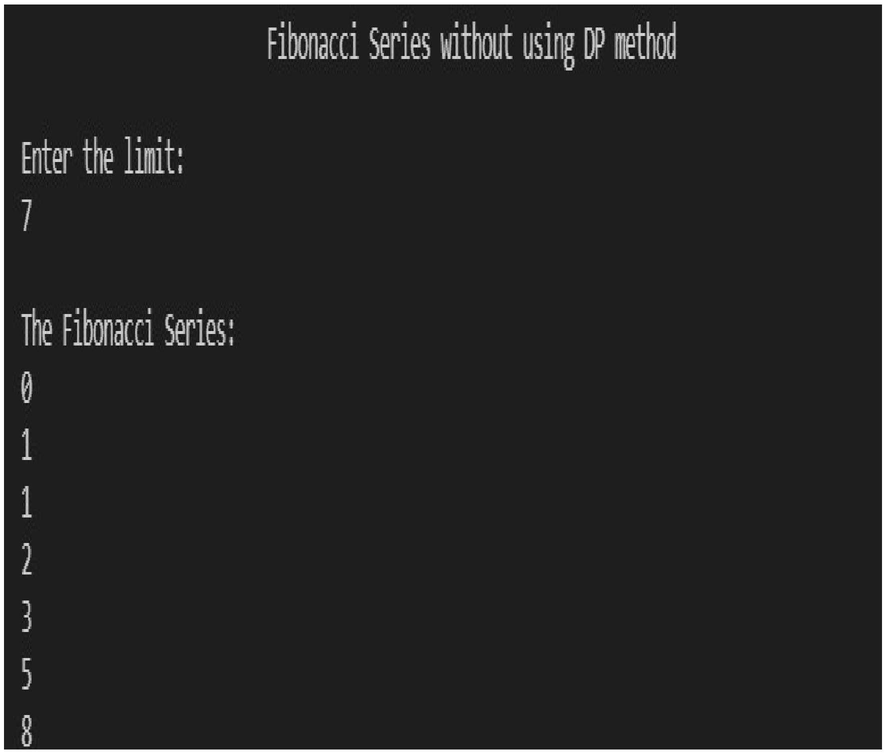
Description: In this series, the next number is the sum of previous two numbers. Without using DP method, the values can be calculated every time at each and every stage of calculation.

Source Code:

```
#include<stdio.h>
#include<conio.h>
int main(int argc, char const *argv[])
{
   int a[10], n, j;
   // clrscr();
   printf("\n\t\t\tFibonacci Series without using DP method\n\n");
   printf("Enter the limit:\n");
   scanf("%d", &n);
   printf("\nThe Fibonacci Series:\n");
   for(int i = 0; i < n; i++)
   {
      printf("%d\n", fin(i));
   }
   return 0;
}
int fin(int n)
{
   if(n==0)
   {
      return 0;
```

```
    }
    else if(n==1)
    {
       return 1;
    }
    else
    {
       return (fin(n-1)+fin(n-2));
    }
 }
```

OUTPUT:



```
                    Fibonacci Series without using DP method

Enter the limit:
7

The Fibonacci Series:
0
1
1
2
3
5
8
```

Program – 07

AIM: Write a program for the implementation of Activity selection problem using Greedy Approach.

Description: The activity selection problem is a mathematical optimization problem. Our first illustration is the problem of scheduling a resource among several challenge activities. We find a greedy algorithm provides a well designed and simple method for selecting a maximum- size set of manually compatible activities.

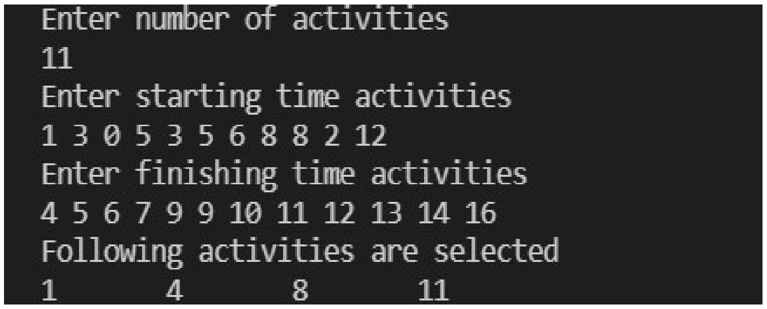
Suppose S = {1, 2....n} is the set of n proposed activities. The activities share resources which can be used by only one activity at a time, e.g., Tennis Court, Lecture Hall, etc. Each Activity "i" has **start time** s_i and a **finish time** f_i, where $s_i \leq f_i$. If selected activity "i" take place meanwhile the half-open time interval $[s_i,f_i)$. Activities i and j are **compatible** if the intervals (s_i, f_i) and $[s_i, f_i)$ do not overlap (i.e. i and j are compatible if $s_i \geq f_i$ or $s_i \geq f_i$). The activity-selection problem chosen the maximum- size set of mutually consistent activities.

Source Code:

```
#include<stdio.h>
#include<conio.h>

int s[100], f[100], n, i, j;
void printMaxActivities()
{
   printf("Following activities are selected\n");
   i = 0;
   printf("%d\t", i+1);
   for(j = 1; j < n; j++)
   {
```

```
            if(s[j] >= f[i])
            {
                printf("%d\t", j+1);
                i = j;
            }
        }
    }

    int main(int argc, char const *argv[])
    {
        printf("Enter number of activities\n");
        scanf("%d", &n);
        printf("Enter starting time activities\n");
        for(int i = 0; i < n; i++)
        {
            scanf("%d", &s[i]);
        }
        printf("Enter finishing time activities\n");
        for(int i = 0; i < n; i++)
        {
            scanf("%d", &f[i]);
        }
        printMaxActivities();
        return 0;
    }
```

OUTPUT:



```
Enter number of activities
11
Enter starting time activities
1 3 0 5 3 5 6 8 8 2 12
Enter finishing time activities
4 5 6 7 9 9 10 11 12 13 14 16
Following activities are selected
1       4       8       11
```

Program – 08

AIM: Write a program to solve travelling salesman problem.

Description: The traveling salesman problems abide by a salesman and a set of cities. The salesman has to visit every one of the cities starting from a certain one (e.g., the hometown) and to return to the same city. The challenge of the problem is that the traveling salesman needs to minimize the total length of the trip.

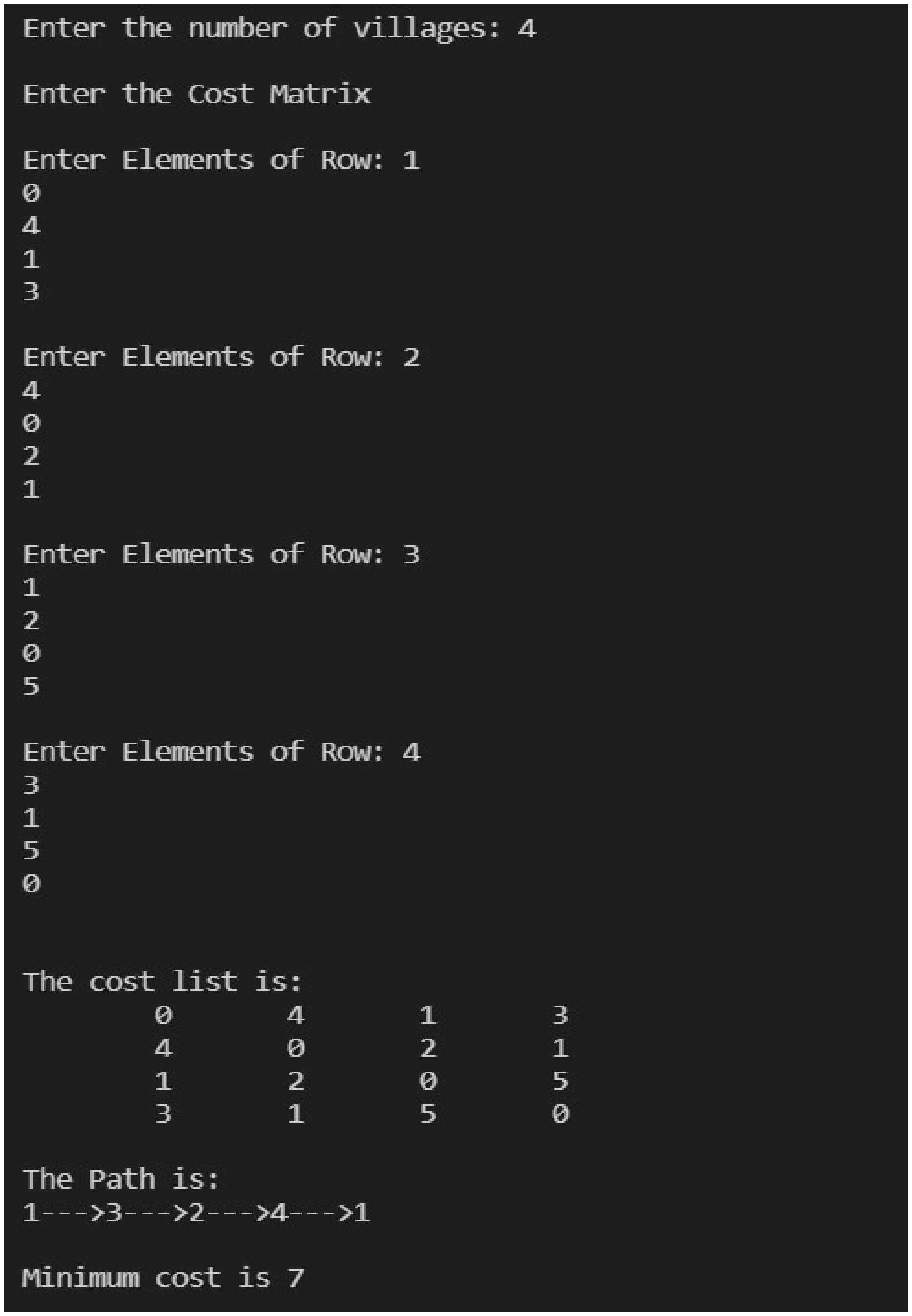
Suppose the cities are x_1 x_2..... x_n where cost c_{ij} denotes the cost of travelling from city x_i to x_j. The travelling salesperson problem is to find a route starting and ending at x_1 that will take in all cities with the minimum cost.

Source Code:

```
#include<stdio.h>

int ary[10][10],completed[10],n,cost=0;

void takeInput()

{

int i,j;

printf("Enter the number of villages: ");

scanf("%d",&n);

printf("\nEnter the Cost Matrix\n");

for(i=0;i < n;i++)

{

printf("\nEnter Elements of Row: %d\n",i+1);

for( j=0;j < n;j++)

scanf("%d",&ary[i][j]);

completed[i]=0;

}
```

```
printf("\n\nThe cost list is:");
for( i=0;i < n;i++)
{
printf("\n");
for(j=0;j < n;j++)
printf("\t%d",ary[i][j]);
}
}

void mincost(int city)
{
int i,ncity;
completed[city]=1;
printf("%d--->",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=ary[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i < n;i++)
{
if((ary[c][i]!=0)&&(completed[i]==0))
if(ary[c][i]+ary[i][c] < min)
{
min=ary[i][0]+ary[c][i];
kmin=ary[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}

int main()
{
takeInput();
printf("\n\nThe Path is:\n");
mincost(0); //passing 0 because starting vertex
printf("\n\nMinimum cost is %d\n ",cost);

return 0;}
```

OUTPUT:



```
Enter the number of villages: 4

Enter the Cost Matrix

Enter Elements of Row: 1
0
4
1
3

Enter Elements of Row: 2
4
0
2
1

Enter Elements of Row: 3
1
2
0
5

Enter Elements of Row: 4
3
1
5
0


The cost list is:
        0       4       1       3
        4       0       2       1
        1       2       0       5
        3       1       5       0

The Path is:
1--->3--->2--->4--->1

Minimum cost is 7
```

Program – 09

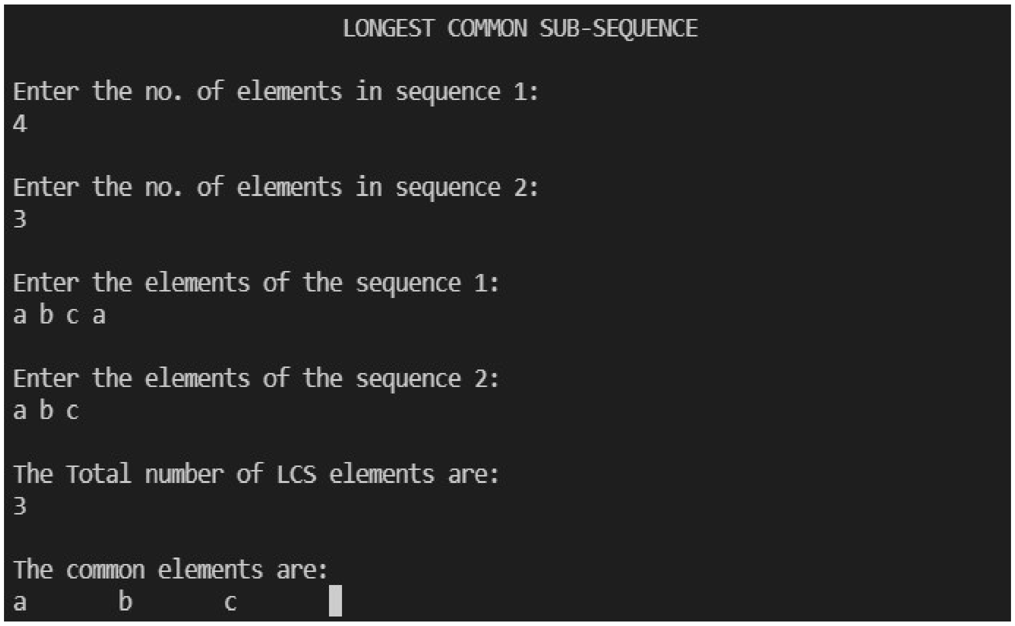
AIM: C program to implement Longest Common Sub-Sequence.

Description: Here longest means that the subsequence should be the biggest one. The common means that some of the characters are common between the two strings. The subsequence means that some of the characters are taken from the string that is written in increasing order to form a subsequence.

Source Code:

```
#include<stdio.h>
#include<conio.h>
char b[10][10];
void lcs(int m,int n,char x[],char y[]);
void printlcs(char b[][10],char x[],int i,int j);
void main()
{
        int m,n,i,j;
        char x[10],y[10],t[10][10];
        // clrscr();
        printf("\n\t\t\t LONGEST COMMON SUB-SEQUENCE\n\n");
        printf("Enter the no. of elements in sequence 1:\n");
        scanf("%d",&m);
        printf("\nEnter the no. of elements in sequence 2:\n");
        scanf("%d",&n);
        printf("\nEnter the elements of the sequence 1:\n");
        for(i=1;i<=m;i++)
        {
                scanf("%s",&x[i]);
        }

          printf("\nEnter the elements of the sequence 2:\n");
        for(j=1;j<=n;j++)
        {
                scanf("%s",&y[j]);
        }
        lcs(m,n,x,y);
        printf("\nThe common elements are:\n");
        printlcs(b,x,m,n);
        getch();
}
void lcs(int m,int n,char x[],char y[])
{
        int c[10][10],i,j;
        for(i=1;i<=m;i++)
        {
                c[i][0]=0;
        }
        for(j=0;j<=n;j++)
        {
                c[0][j]=0;
        }
        for(i=1;i<=m;i++)
        {
                for(j=1;j<=n;j++)
                {
                    if(x[i]==y[j])
                    {
                        c[i][j]=c[i-1][j-1]+1;
                        b[i][j]='a';
                    }
                    else if(c[i-1][j]>=c[i][j-1])
                    {
                        c[i][j]=c[i-1][j];
                        b[i][j]='b';
                    }
                    else
                    {
                        c[i][j]=c[i][j-1];
                        b[i][j]='c';
                    }
                }
        }
        printf("\nThe Total number of LCS elements are:\n");
        printf("%d\n",c[m][n]);
}
void printlcs(char b[][10],char x[],int i,int j)
{
        if(i==0 && j==0)
        {

                return;
        }
        if(b[i][j]=='a')
        {
            printlcs(b,x,i-1,j-1);
            printf("%c\t",x[i]);
        }
        else if(b[i][j]=='b')
        {
            printlcs(b,x,i-1,j);
        }
        else
        {
            printlcs(b,x,i,j-1);
        }
}
```

OUTPUT:



```
                    LONGEST COMMON SUB-SEQUENCE

Enter the no. of elements in sequence 1:
4

Enter the no. of elements in sequence 2:
3

Enter the elements of the sequence 1:
a b c a

Enter the elements of the sequence 2:
a b c

The Total number of LCS elements are:
3

The common elements are:
a       b       c
```

Program – 11

AIM: Write a program to implement MST of given undirected graph using Kruskal's algorithm.

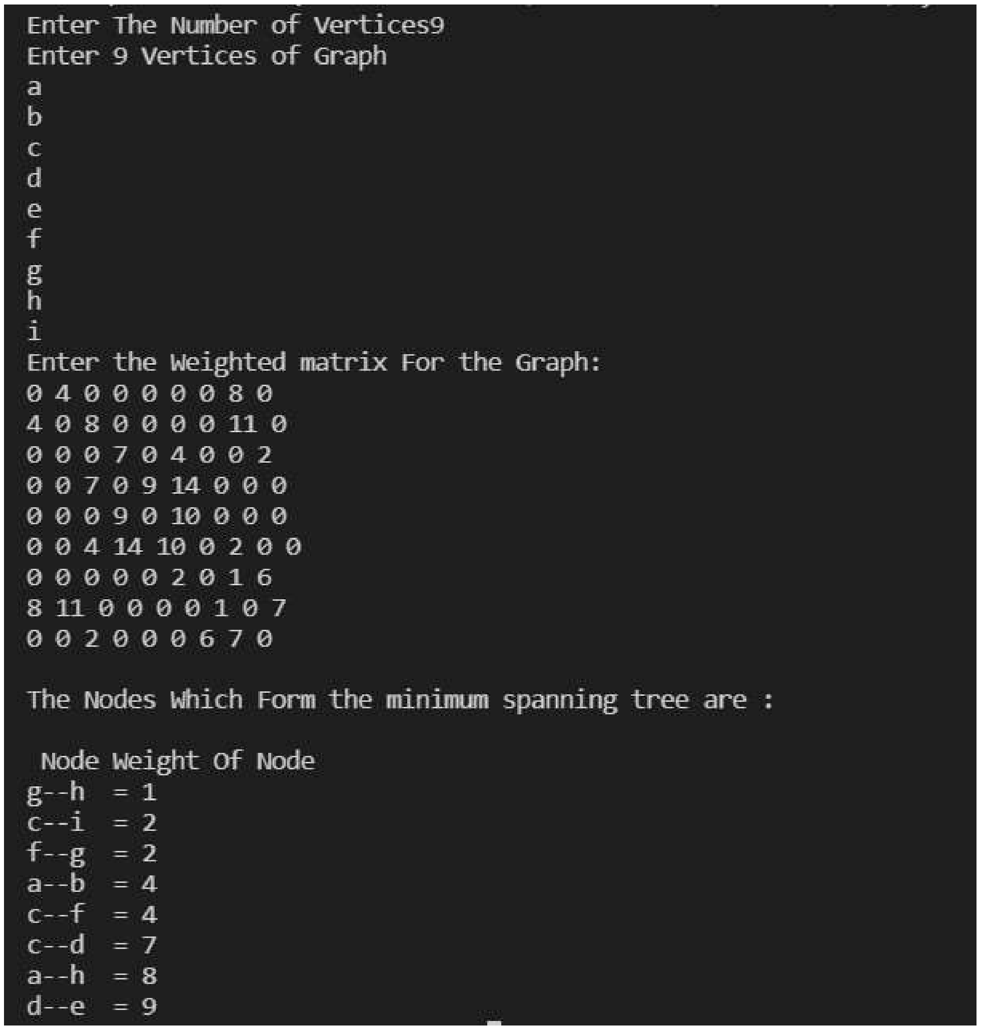
Description: Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest. It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component).

Source Code:

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

char vertex[10];
int weight[10][10], size, set_index[10], min, i, j, u, v, k, range = 1, temp;

void build_graph()
{
   printf("Enter The Number of Vertices");
   scanf("%d", &size);
   printf("Enter %d Vertices of Graph \n", size);
   for (i = 0; i < size; i++)
   {
      fflush(stdin);
      scanf("%c", &vertex[i]);
```

```
        set_index[i] = i;
    }
    printf("Enter the Weighted matrix For the Graph:\n");
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            scanf("%d", &weight[i][j]);
        }
    }
}

void kruskal()
{
    for (k = 0; k < size; k++)
    {
        min = INT_MAX;
        for (i = 0; i < size; i++)
        {
            for (j = 0; j < size; j++)
            {
                if (weight[i][j] < min && weight[i][j] >= range && set_index[i] !=
set_index[j])
                {
                    min = weight[i][j];
                    u = i;
                    v = j;
                }
            }
        }
        range = min;
        if (set_index[u] != set_index[v])
        {
            temp = set_index[v];
            for (i = 0; i < size; i++)
            {
                if (set_index[i] == temp)
                {
                    set_index[i] = set_index[u];
                }
            }
            printf("%c--%c  = %d\n", vertex[u], vertex[v], weight[u][v]);
        }
    }
}
int main()
{
    build_graph();
    printf("\nThe Nodes Which Form the minimum spanning tree are : \n\n Node
Weight Of Node\n");
    kruskal();
    return 0;
}
```

OUTPUT:



```
Enter The Number of Vertices9
Enter 9 Vertices of Graph
a
b
c
d
e
f
g
h
i
Enter the Weighted matrix For the Graph:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 0 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

The Nodes Which Form the minimum spanning tree are :

 Node Weight Of Node
g--h  = 1
c--i  = 2
f--g  = 2
a--b  = 4
c--f  = 4
c--d  = 7
a--h  = 8
d--e  = 9
```

Program – 10

AIM: Write a program to implement MST of given undirected graph using Prim's algorithm.

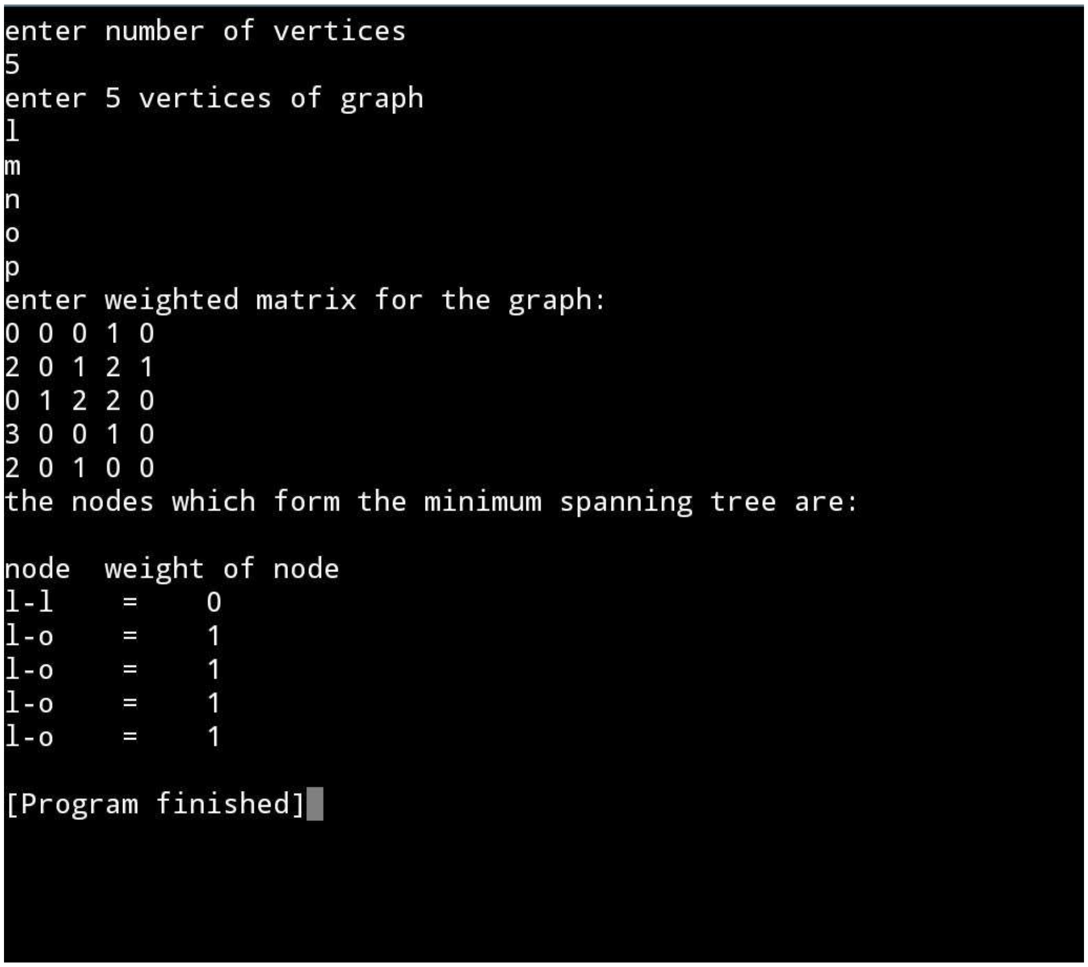
Description: Prim's (also known as Jarník's) algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Source Code:

```
#include<stdio.h>
#include<conio.h>
#include<limits.h>
#include<stdlib.h>
char vertex[10];
int weight[10][10],key[10],size,visited[10],parent[10],min,min_index,i,j,u;
void build_graph()
{
        printf("enter number of vertices\n");
        scanf("%d",&size);
        printf("enter %d vertices of graph\n",size);
        for(i=0;i<size;i++)
        {
                fflush(stdin);
                scanf("%c",&vertex[i]);
        }
        printf("enter weighted matrix for the graph:\n");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        scanf("%d",&weight[i][j]);
                }
        }
}
int extract_min()
{
        min=INT_MAX;
        for(i=0;i<size;i++)
        {
                if(!visited[i]&&key[i]<min)
                {

min=key[i];
                        min_index=i;
                }
        }
        return min_index;
}
void prims()
{
        for(i=0;i<size;i++)
        {
                key[i]=INT_MAX;
                visited[i]=0;
                parent[i]=NULL;
        }
        key[0]=0;
        for(j=0;j<size;j++)
        {
                u=extract_min();
                visited[u]=1;
                printf("%c-%c    =
%d\n",vertex[parent[u]],vertex[u],weight[parent[u]][u]);
                for(i=0;i<size;i++)
                {
                        if(!visited[i] && weight[u][i]!=0)
                        {
                                if(key[i]>weight[u][i])
                                {
                                        key[i]=weight[u][i];
                                        parent[i]=u;
                                }
                        }
                }
        }
}
void main()
{
        build_graph();
        printf("the nodes which form the minimum spanning tree are:\n\nnode
weight of node\n");
```

```
        prims();
        getch();
}
```

OUTPUT:



```
enter number of vertices
5
enter 5 vertices of graph
l
m
n
o
p
enter weighted matrix for the graph:
0 0 0 1 0
2 0 1 2 1
0 1 2 2 0
3 0 0 1 0
2 0 1 0 0
the nodes which form the minimum spanning tree are:

node  weight of node
l-l    =    0
l-o    =    1
l-o    =    1
l-o    =    1
l-o    =    1

[Program finished]
```

Program – 14

AIM: Implement All-Pairs Shortest Paths Problem using Floyd's-Warshall Algorithm.

Description: The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

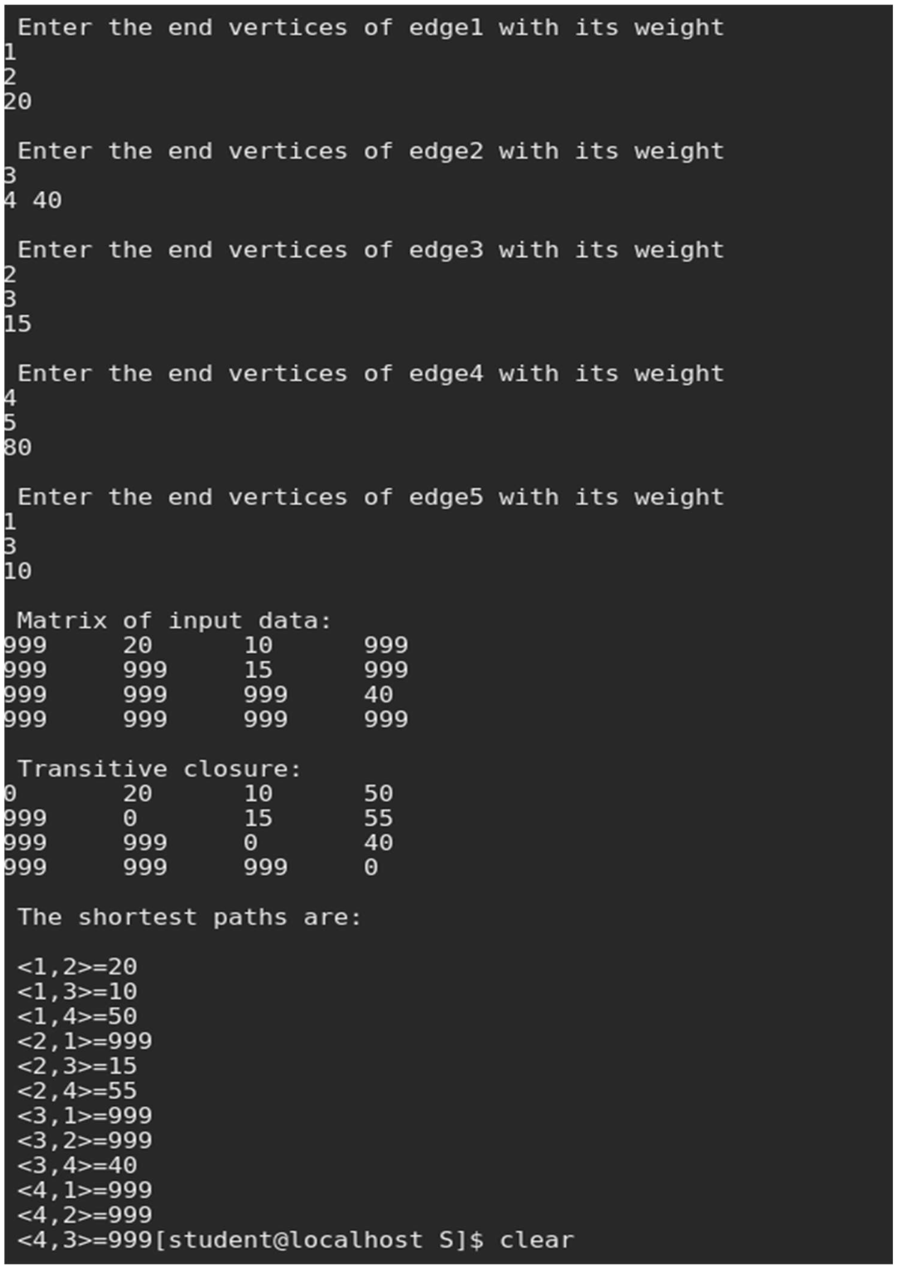
Source Code:

```
#include<stdio.h>
#define V 4
#define INF 99999
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])
{
   printf ("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)
                printf("%7s", "INF");
            else
                printf ("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int graph[V][V] = { {0,   5,  INF, 10},
                        {INF, 0,   3, INF},
                        {INF, INF, 0,   1},
                        {INF, INF, INF, 0}
                      };

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

OUTPUT:

m



```
 Enter the end vertices of edge1 with its weight
1
2
20

 Enter the end vertices of edge2 with its weight
3
4 40

 Enter the end vertices of edge3 with its weight
2
3
15

 Enter the end vertices of edge4 with its weight
4
5
80

 Enter the end vertices of edge5 with its weight
1
3
10

 Matrix of input data:
999     20      10      999
999     999     15      999
999     999     999     40
999     999     999     999

 Transitive closure:
0       20      10      50
999     0       15      55
999     999     0       40
999     999     999     0

 The shortest paths are:

 <1,2>=20
 <1,3>=10
 <1,4>=50
 <2,1>=999
 <2,3>=15
 <2,4>=55
 <3,1>=999
 <3,2>=999
 <3,4>=40
 <4,1>=999
 <4,2>=999
 <4,3>=999[student@localhost S]$ clear
```

Program – 12

AIM: From a given vertex in a weighted connected graph,find shortest paths to other vertices using Dijkistra's algorithm.

Description: 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 a given source as a root. We maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, we find a vertex that is in the other set (set of not yet included) and has a minimum distance from the source.

Source Code:

```
#include <iostream>
using namespace std;
#include <limits.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
   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;
}
void printSolution(int dist[])
{
   cout <<"Vertex \t Distance from Source" << endl;
   for (int i = 0; i < V; i++)
      cout  << i << " \t\t"<<dist[i]<< endl;
}
void dijkstra(int graph[V][V], int src)
{
   int dist[V]; // The output array.  dist[i] will hold the shortest
   bool sptSet[V];

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

   dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
      int u = minDistance(dist, sptSet);
```

```
        sptSet[u] = true;

        for (int v = 0; v < V; 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];
    }

    printSolution(dist);
}

int main()
{
    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;
}
```

OUTPUT:

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

Program – 13

AIM: Implement Bellman-Ford Algorithm to find Shortest Path from a source.

Description: The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.[1] It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel (1955), but is instead named after Richard Bellman and Lester Ford Jr., who published it in 1958 and 1956, respectively.[2] Edward F. Moore also published a variation of the algorithm in 1959, and for this reason it is also sometimes called the **Bellman–Ford**

Source Code:

```
#include <iostream>

#include <vector>

#include <iomanip>

#include <climits>

using namespace std;

struct Edge {

   int source, dest, weight;

};

void printPath(vector<int> const &parent, int vertex)

{

   if (vertex < 0) {

      return;

   }
```

```
    printPath(parent, parent[vertex]);
    cout << vertex << " ";
}
void bellmanFord(vector<Edge> const &edges, int source, int N)
{
    vector<int> distance (N, INT_MAX);
    distance[source] = 0;
    vector<int> parent (N, -1);
    int u, v, w, k = N;
    while (--k)
    {
        for (Edge edge: edges)
        {
            u = edge.source;
            v = edge.dest;
            w = edge.weight;
            if (distance[u] != INT_MAX && distance[u] + w < distance[v])
            {
                distance[v] = distance[u] + w;
                parent[v] = u;
            }
        }
    }
    for (Edge edge: edges)
    {
        u = edge.source;
        v = edge.dest;
        w = edge.weight;
        if (distance[u] != INT_MAX && distance[u] + w < distance[v])
        {
            cout << "Negative-weight cycle is found!!";
            return;
        }
    }
    for (int i = 0; i < N; i++)
    {
        cout << "The distance of vertex " << i << " from the source is "
             << setw(2) << distance[i] << ". Its path is [ ";
        printPath(parent, i); cout << "]" << endl;
    }
}
int main()
{
    vector<Edge> edges =
    {
        // `(x, y, w)` —> edge from `x` to `y` having weight `w`
        { 0, 1, -1 }, { 0, 2, 4 }, { 1, 2, 3 }, { 1, 3, 2 },
        { 1, 4, 2 }, { 3, 2, 5 }, { 3, 1, 1 }, { 4, 3, -3 }
    };
    int N = 5;
    int source = 0;
    bellmanFord(edges, source, N);
    return 0;
}
```

OUTPUT:

```
The distance of vertex 0 from the source is 0. Its path is [ 0 ]
The distance of vertex 1 from the source is -1. Its path is [ 0 1
]
The distance of vertex 2 from the source is 2. Its path is [ 0 1
2 ]
The distance of vertex 3 from the source is -2. Its path is [ 0 1
4 3 ]
The distance of vertex 4 from the source is 1. Its path is [ 0 1
4 ]
```

Program – 04

AIM: Linear Search

Description: Linear search is a sequential searching algorithm where we start from one end and check every element of the list until the desired element is found. It is the simplest searching algorithm.

Source Code:

```
#include <stdio.h>
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = search(arr, n, x);
    (result == -1)
  ? printf("Element is not present in array")
     : printf("Element is present at index %d", result);
 return 0;
}
```

OUTPUT:

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

PROGRAM-05

AIM: Binary Search

Description: Binary Search is a searching algorithm for finding an element's position in a sorted array.In this approach, the element is always searched in the middle of a portion of an array.

Source Code:

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l) {
      int mid = l + (r - l) / 2;
      if (arr[mid] == x)
         return mid;
      if (arr[mid] > x)
         return binarySearch(arr, l, mid - 1, x);
      return binarySearch(arr, mid + 1, r, x);
   }
   return -1;
}
int main(void)
{
   int arr[] = { 2, 3, 4, 10, 40 };
   int n = sizeof(arr) / sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n - 1, x);
```

```
    (result == -1)
        ? printf("Element is not present in array")
        : printf("Element is present at index %d", result);
    return 0;
}
```

OUTPUT:

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