Program 2

Design and implement C Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
#include <stdio.h>

#include <limits.h>

#define V_MAX 100 // Maximum number of vertices

// Function to find the vertex with the minimum key value, from the set of vertices not yet included in the MST

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

  int min = INT_MAX, min_index;

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

    if (mstSet[v] == 0 && key[v] < min)

      min = key[v], min_index = v;

  return min_index;

}

// Function to print the constructed MST stored in parent[]

void printMST(int parent[], int n, int graph[V_MAX][V_MAX], int V) {

  printf("Edge   Weight\n");

  for (int i = 1; i < V; i++)

    printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);

}


// Function to construct and print MST for a graph represented using adjacency matrix representation

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

  int parent[V_MAX]; // Array to store constructed MST

  int key[V_MAX]; // Key values used to pick minimum weight edge in cut

  int mstSet[V_MAX]; // To represent set of vertices not yet included in MST


  // Initialize all keys as INFINITE, mstSet[] as 0

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

    key[i] = INT_MAX, mstSet[i] = 0;
```

```
  // Always include first 1st vertex in MST. Make key 0 so that this vertex is picked as the first vertex
  key[0] = 0;
  parent[0] = -1; // First node is always the 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, V);

    // Add the picked vertex to the MST set
    mstSet[u] = 1;

    // Update key value and parent index of the adjacent vertices of the picked vertex
    // Consider only those vertices which are not yet included in the MST
    for (int v = 0; v < V; v++)
      if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
        parent[v] = u, key[v] = graph[u][v];
  }

  // Print the constructed MST
  printMST(parent, V, graph, V);
}

int main() {
  int V, E;
  printf("Enter the number of vertices and edges: ");
  scanf("%d %d", &V, &E);

  // Create the graph as an adjacency matrix
  int graph[V_MAX][V_MAX];
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
```

```
            graph[i][j] = 0; // Initialize the graph with 0s
        }
    }

    // Prompt the user to enter the source vertex, destination vertex, and weight for each edge
    printf("Enter the source vertex, destination vertex, and weight for each edge:\n");
    for (int i = 0; i < E; i++) {
        int source, dest, weight;
        scanf("%d %d %d", &source, &dest, &weight);
        graph[source][dest] = weight;
        graph[dest][source] = weight; // Since the graph is undirected
    }

    // Print the MST using Prim's algorithm
    primMST(graph, V);

    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gedit 2.c
student@lenovo-ThinkCentre-M900:~$ gcc 2.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices and edges: 5
7
Enter the source vertex, destination vertex, and weight for each edge:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Program 3

3.a. Design and implement C Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

PROGRAM:

```
#include<stdio.h>
   int min(int,int);
void floyds(int p[10][10],int n) {
       int i,j,k;
       for (k=1;k<=n;k++)
         for (i=1;i<=n;i++)
         for (j=1;j<=n;j++)
         if(i==j)
           p[i][j]=0;                          else
           p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b) {
       if(a<b)
       return(a); else
       return(b);
}
void main() {
       int p[10][10],w,n,e,u,v,i,j;

       printf("\n Enter the number of vertices:");
       scanf("%d",&n);
       printf("\n Enter the number of edges:\n");
       scanf("%d",&e);
       for (i=1;i<=n;i++) {
               for (j=1;j<=n;j++)
                 p[i][j]=999;
       }
```

```
        for (i=1;i<=e;i++) {
                printf("\n Enter the end vertices of edge%d with its weight \n",i);
                scanf("%d%d%d",&u,&v,&w);
                p[u][v]=w;
        }
        printf("\n Matrix of input data:\n");
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++)
                  printf("%d \t",p[i][j]);
                printf("\n");
        }
        floyds(p,n);
        printf("\n Transitive closure:\n");
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++)
                  printf("%d \t",p[i][j]);
                printf("\n");
        }
        printf("\n The shortest paths are:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++) {
                if(i!=j)
                   printf("\n <%d,%d>=%d",i,j,p[i][j]);
        }

   }
```

OUTPUT:

Enter the number of vertices:4

Enter the number of edges:

5

Enter the end vertices of edge1 with its weight

1 3 3

Enter the end vertices of edge2 with its weight

2 1 2

Enter the end vertices of edge3 with its weight

3 2 7

Enter the end vertices of edge4 with its weight

3 4 1

Enter the end vertices of edge5 with its weight

4 1 6

Matrix of input data:

999	999	3	999
2	999	999	999
999	7	999	1
6	999	999	999

Transitive closure:

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

The shortest paths are:

<1,2>=10

<1,3>=3

<1,4>=4

<2,1>=2

<2,3>=5

<2,4>=6

<3,1>=7

<3,2>=7

<3,4>=1

<4,1>=6

<4,2>=16

3b.Design and implement C Program to find the transitive closure using Warshal's algorithm.

PROGRAM:

```
#include<stdio.h>
#include<math.h>
int max(int, int);
void warshal(int p[10][10], int n) {
  int i, j, k;
  for (k = 1; k <= n; k++)
    for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
        p[i][j] = max(p[i][j], p[i][k] && p[k][j]);
}

int max(int a, int b) {

  ;

  if (a > b)
    return (a);
  else
    return (b);
}

void main() {
  int p[10][10] = { 0 }, n, e, u, v, i, j;
```

```
printf("\n Enter the number of vertices:");
scanf("%d", &n);
printf("\n Enter the number of edges:");
scanf("%d", &e);
for (i = 1; i <= e; i++) {
   printf("\n Enter the end vertices of edge %d:", i);
   scanf("%d%d", &u, &v);
   p[u][v] = 1;
}
printf("\n Matrix of input data: \n");
for (i = 1; i <= n; i++) {
   for (j = 1; j <= n; j++)
      printf("%d\t", p[i][j]);
   printf("\n");
}
warshal(p, n);
printf("\n Transitive closure: \n");
for (i = 1; i <= n; i++) {
   for (j = 1; j <= n; j++)
      printf("%d\t", p[i][j]);
   printf("\n");
}
 }
```

OUTPUT:

```
gedit 3b.c
gcc 3b.c
./a.out
 Enter the number of vertices:5
 Enter the number of edges:11

 Enter the end vertices of edge 1:1 1
```

Enter the end vertices of edge 2:1 4

Enter the end vertices of edge 3:3 2

Enter the end vertices of edge 4:3 3

Enter the end vertices of edge 5:3 4

Enter the end vertices of edge 6:4 2

Enter the end vertices of edge 7:4 4

Enter the end vertices of edge 8:5 2

Enter the end vertices of edge 9:5 3

Enter the end vertices of edge 10:5 4

Enter the end vertices of edge 11:5 5

Matrix of input data:

1	0	0	1	0
0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	1

Transitive closure:

1	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	1

Program 4

4. Design and implement C Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

```
#include <stdio.h>

#include <stdbool.h>

#include <limits.h>


#define MAX_VERTICES 10 // Maximum number of vertices

#define INF INT_MAX


// A function to find the vertex with the minimum distance value, from the set of vertices not yet
included in the shortest path tree

int minDistance(int dist[], bool sptSet[], int V) {

   int min = INF, 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

void printSolution(int dist[], int V) {

   printf("Vertex \t\t Distance from Source\n");

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

      printf("%d \t\t %d\n", i, dist[i]);

}

// Dijkstra's algorithm for adjacency matrix representation of the graph

void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int V) {

   int dist[MAX_VERTICES];  // The output array. dist[i] will hold the shortest distance from src to i

   bool sptSet[MAX_VERTICES];  // sptSet[i] will be true if vertex i is included in the shortest path
tree


   // Initialize all distances as INFINITE and sptSet[] as false
```

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

    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist, V);
}
// Driver code
int main() {
    int V, E;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);

    int graph[MAX_VERTICES][MAX_VERTICES] = {{0}};

    printf("Enter the source vertex, destination vertex, and weight for each edge:\n");
    for (int i = 0; i < E; i++) {
        int source, dest, weight;
        scanf("%d %d %d", &source, &dest, &weight);
        graph[source][dest] = weight;
        graph[dest][source] = weight; // Assuming undirected graph
    }
```

```
    dijkstra(graph, 0, V);

    return 0;

}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gedit 4.c
student@lenovo-ThinkCentre-M900:~$ gcc 4.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices: 5
Enter the number of edges: 7
Enter the source vertex, destination vertex, and weight for each edge:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Vertex          Distance from Source
0               0
1               2
2               5
3               6
4               7
```