EXPERIMENT NO. 10

Author: Harsheet Chordiya

Roll No.: 37

Sem and Section: 3 CSEB

Source file: expt10.c

Date: 21/02/2021

==========

Aim: To demonstrate shortest path algorithms – Floyd-Warshall algorithm and Dijkstra's algorithm.

Problem Statement: Create an arbitrary graph with a minimum of 8 nodes [use the routines developed in Experiment No. 09].

Implement the routines to –

(1) Find shortest path matrix using Warshall's algorithm
(2) Find shortest path between vertices Dijkstra's algorithm.

Write a menu driven program to implement the solution.You must include other utility functions in the menu.

==========

THEORY

==========

I. Definitions :

- **Floyd-Warshall**

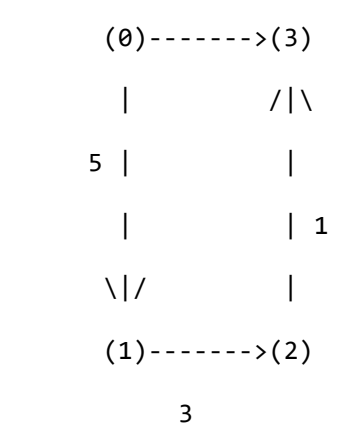
The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

Example:

Input:

```
       graph[][] = { {0,   5,  INF, 10},
                     {INF,  0,  3,  INF},
                     {INF, INF, 0,   1},
                     {INF, INF, INF, 0} }
```

which represents the following graph



Note that the value of graph[i][j] is 0 if i is equal to j

And graph[i][j] is INF (infinite) if there is no edge from vertex i to j.

Output:

Shortest distance matrix

```
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
```

Dijkstra's:

Dijkstra algorithm is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes.

II. Algorithms :

- **Floyd-Warshall**

Pre-Conditions:

i. Edge cost should be non-negative.
ii. Weighted directed graph

Required Data Structures:

i. A 2-D array, DIST[][] representing the distance of the vertex from any source vertex.
ii. A 2-D array, NEXT[][] representing the vertex which will be reached next from the currrent vertex.

Algorithm :

```
n = no of vertices

A = matrix of dimension n*n

for k = 1 to n

    for i = 1 to n

        for j = 1 to n

            Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])

return A
```

➢ Dijkstra's

Pre-Conditions:

i. Edge cost should be non-negative.
ii. Weighted directed graph

Required Data Structures:

i. A Queue [an array Visited[] to keep track of already vertices].
ii. Also a minHeap or Fibonacci Heap may also be used.
iii. Two 1-D arrays of size = |V|. Array Length[] will store distance of the vertex from source.

iv. Array Parent[] will indicate the parent vertex of the current vertex.

Algorithm :

```
function dijkstra(G, S)

    for each vertex V in G

        distance[V] <- infinite

        previous[V] <- NULL

        If V != S, add V to Priority Queue Q

    distance[S] <- 0


    while Q IS NOT EMPTY

        U <- Extract MIN from Q

        for each unvisited neighbour V of U

            tempDistance <- distance[U] + edge_weight(U, V)

            if tempDistance < distance[V]

                distance[V] <- tempDistance

                previous[V] <- U

    return distance[], previous[]
```

==========

PROGRAM

==========

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

#define MAX 10
#define NF 999

int minDistance(int dist[], int visited[], int vertices);
void printDijkstraState(int dist[], int visited[],int prev[], int vertices);
int insert(int Q[], int *front, int *rear, int key);
void displayQ(int Q[], int front, int rear);
void savepath(int vertex,int prev[],int path[],int start,int *front,int *rear);
void initQueue(int *front, int *rear);
void printpath(int vertices,int prev[],int start);
void showAdjacencyMatrix(int graph[][MAX], int vertices);
void spDijkstra(int Graph[][MAX], int vertices, int start);
int createGraph(int graph[][MAX]);
void initAdjacencyMatrix(int graph[][MAX]);
void spfloyd(int Graph[][MAX], int vertices, int start );


int main() {
  int original[MAX][MAX],graph[MAX][MAX], i, j, vertices, u,key;
  vertices = 6;

  initAdjacencyMatrix(original);
  initAdjacencyMatrix(graph);
```

```
vertices = createGraph(original);
do
{
  for(i=0;i<vertices;i++)
  {
      for(j=0;j<vertices;j++)
      {
          graph[i][j]=original[i][j];
      }
  }
  printf("\nEnter the method you want to implement : ");
  printf("\n 1. Dijkshtra algorithm   2.Warshall Floyd   0.Exit");
  printf("\ncode::");
  scanf("%d",&key);

  switch (key)
  {
      case 0: printf("\nThank you");
              break;

      case 1: printf("\nenter the source vertex :");
              scanf("%d",&u);

              spDijkstra(graph, vertices, u);
              break;

      case 2: printf("\nenter the source vertex :");
              scanf("%d",&u);
              spfloyd(graph, vertices, u);
              break;
```

```
        default:printf("\ninvalid choice....\n");
                break;
    }
  }while(key!=0);



  return 0;
}

// Declaration

int minDistance(int dist[], int visited[], int vertices)
{
  int min = NF, minPos, i;
  for(i = 0; i < vertices; i++)
  if(visited[i] == 0 && dist[i] <= min)
  min = dist[i], minPos = i;
  return minPos;
}

void printDijkstraState(int dist[], int visited[],int prev[], int vertices)
{
    int i,j;

    printf("\t\t      v |");
    for (i = 0; i < vertices; i++)
        printf("%6d", i);
    printf("\n");
```

```
    printf("\t\t   prev |");
    for (i = 0; i < vertices; i++)
        printf("%6d", prev[i]);
    printf("\n");

    printf("\t\t   dist |");
    for (i = 0; i < vertices; i++)
        printf("%6d", dist[i]);
    printf("\n");

    printf("\t\tvisited |");
    for (i = 0; i < vertices; i++)
        printf("%6d", visited[i]);
    printf("\n\n");

}

int insert(int Q[], int *front, int *rear, int key)
{
    if (*rear == MAX - 1)
        return 999999; // Full
    *rear = *rear + 1;
    Q[(*rear)] = key;
    if (*front == -1) // initial case when initQueue()
        *front = 0;
}

void displayQ(int Q[], int front, int rear)
{
```

```
    int i;
    if(rear != -1)
    {
        printf("\n");
        for(i=rear-1; i>=front; i--)
            printf("%4d-->", Q[i]);
    }
    else
    printf("\n\tQueue is Empty....\n");
}

void savepath(int vertex,int prev[],int path[],int start,int *front,int *rear)
{
    if(prev[vertex]==-1)
    {
        insert(path,front,rear,0);
        return;
    }
    insert(path,front,rear,prev[vertex]);
    savepath(prev[vertex],prev,path,start,front,rear);
}

void initQueue(int *front, int *rear)
{
    *front = -1;
    *rear = -1;
}

void printpath(int vertices,int prev[],int start)
{
```

```
    int i,path[MAX],front=-1,rear=-1;
    printf("\n\n");
    for(i=0;i<vertices;i++)
    {
        int j=0;
        printf("%d. ",i);
        savepath(i,prev,path,start,&front,&rear);
        displayQ(path,front,rear);
        printf("%4d",i);
        printf("\n\n");
        initQueue(&front,&rear);
    }

}

void showAdjacencyMatrix(int graph[][MAX], int vertices)
{
    int i, j;

    printf("\n\n\t\t u|v |");
    for (i = 0; i < vertices; i++)
        printf("%4d", i);
    printf("\n");

    printf("\t\t-------");
    for (i = 0; i < vertices; i++)
        printf("----");
    printf("\n");

    for (i = 0; i < vertices; i++)
```

```
    {
        printf("\t\t%4d  |", i);
        for (j = 0; j < vertices; j++)
            printf("%4d", graph[i][j]);
        printf("\n");
    }
    printf("\n");
}

void spDijkstra(int Graph[][MAX], int vertices, int start)
{
  int dist[MAX],visited[MAX],prev[MAX],i,u,v;

  for(i=0;i<vertices;i++)
  {
    prev[i]=-1;
    dist[i]=NF;
    visited[i]=0;
  }

  dist[start]=0;
  for ( i = 0; i < vertices; i++)
  {
    u=minDistance(dist,visited,vertices);
    visited[u]=1;

    for ( v = 0; v < vertices; v++)
    {
      if(!visited[v] && Graph[u][v] && dist[u] + Graph[u][v]<dist[v])
      {
```

```
        prev[v]=u;
        dist[v]=dist[u]+Graph[u][v];
      }

    }
    printDijkstraState( dist,  visited, prev,  vertices);

  }

  // Printing the distance

  for (i = 0; i < vertices; i++)
    if (i != start)
    {
      printf("\nDistance from source to %d: %d", i, dist[i]);
    }
    printpath(vertices,prev,start);
}
int createGraph(int graph[][MAX])
{
    int i, j, vCnt = 0,weight;
    int u, v, vertices, type;

    printf("\n\tGraph Creation [Undirected/Directed]...\n");

    printf("\t\tType of Graph [0: UnDirected] := ");
    scanf("%d", &type);

    if (type != 0)
        type = 1;
```

```
do
{
    printf("\t\tHow many vertices [upto %d vertices]?? ", MAX);
    scanf("%d", &vertices);
} while (vertices < 1 || vertices > MAX);

printf("\n\t...\n");

printf("\n\tVerices starts at 0 and terminates at %d\n", vertices - 1);
printf("\t\tVertex ID of -1 terminates Input\n");
printf("\n\tEnter Existing Edges in the Graph\n\n");
printf("\t\t-------------------------------------------------------\n");
printf("\t\tEdge#   uVertex#   vVertex#   Distance      Remark\n");
printf("\t\t-------------------------------------------------------\n");

do
{
    printf("\t\t%5d", vCnt + 1);
    scanf("%d%d%d", &u, &v, &weight);

    if ((u != -1 || v != -1) && u < vertices && v < vertices)
    {
        if (graph[u][v] == NF)
        {
            if (type)
                graph[u][v] = weight;
            else
                graph[u][v] = graph[v][u] = weight;
```

```
                printf("\t\t\t\t\t\t\t\tEdge Created\n");
            }
            else
                printf("\t\t\t\t\t\t\t\tEdge Exists\n");
        }
        else
            printf("\t\t\t\t\t\t\t\tInvalid Edge\n");
        vCnt++;
    } while (u != -1 || v != -1);
    return vertices;

    for(i = 0; i < MAX; i++)
    if(graph[i][i] == NF)
        graph[i][i] = 0;
    return vertices;
}

void initAdjacencyMatrix(int graph[][MAX])
{
    int i, j;
    for (i = 0; i < MAX; i++)
        for (j = 0; j < MAX; j++)
            graph[i][j] = NF;
}

void spfloyd(int Graph[][MAX], int vertices, int start )
{
    int i,j,k,pathmatrix[MAX][MAX];

    for (i = 0; i < vertices; i++)
```

```
  {
      for (j = 0; j < vertices; j++)
      {
          Graph[i][j] = Graph[i][j];
          pathmatrix[i][j]=j;
      }
  }

showAdjacencyMatrix(Graph,vertices);

// Adding vertices individually
for (k = 0; k < vertices; k++)
{
  for (i = 0; i < vertices; i++)
  {
    for (j = 0; j < vertices; j++)
    {
      if(i==j)
      {
          continue;
      }
      if (Graph[i][k] + Graph[k][j] < Graph[i][j])
        {
          Graph[i][j] = Graph[i][k] + Graph[k][j];
          pathmatrix[i][j]=pathmatrix[i][k];
        }
    }
  }
  printf("\nthe matrix is :\n");
  showAdjacencyMatrix(Graph,vertices);
```

```
    printf("\nVErtex matrix:\n");
    showAdjacencyMatrix(pathmatrix,vertices);
    printf("\n\n");
  }
   printf("\nthe final matrix is :\n");
   showAdjacencyMatrix(Graph,vertices);
   printf("\nVertex matrix:\n");
   showAdjacencyMatrix(pathmatrix,vertices);
}
```

```
================================================================================
```

EXECUTION-TRAIL

```
================================================================================
Graph Creation [Undirected/Directed]...
                Type of Graph [0: UnDirected] := 0
                How many vertices [upto 10 vertices]?? 4


        ...


        Verices starts at 0 and terminates at 3
                Vertex ID of -1 terminates Input


        Enter Existing Edges in the Graph


                ---------------------------------------------------
                Edge#   uVertex#   vVertex#   Distance      Remark
                ---------------------------------------------------
                    10
0
1
                                                              Edge Created
                    21
2
2
                                                              Edge Created
                    33
4
4
                                                              Invalid Edge
                    4
```

```
-1
-1
-1
                                                                    Invalid Edge


Enter the method you want to implement :
 1. Dijkshtra algorithm   2.Warshall Floyd   0.Exit
code::1


enter the source vertex :2
                      v |     0     1     2     3
                   prev |    -1     2    -1    -1
                   dist |   999     2     0   999
                visited |     0     0     1     0

                      v |     0     1     2     3
                   prev |    -1     2    -1    -1
                   dist |   999     2     0   999
                visited |     0     1     1     0

                      v |     0     1     2     3
                   prev |    -1     2    -1    -1
                   dist |   999     2     0   999
                visited |     0     1     1     1

                      v |     0     1     2     3
                   prev |    -1     2    -1    -1
                   dist |   999     2     0   999
                visited |     1     1     1     1
```

```
Distance from source to 0: 999

Distance from source to 1: 2

Distance from source to 3: 999


0.

   0


1.

   2-->   1


2.

   2


3.

   3



Enter the method you want to implement :

 1. Dijkshtra algorithm   2.Warshall Floyd   0.Exit

code::2


enter the source vertex :2



                u|v |   0   1   2   3

               ----------------------

                  0  |   1 999 999 999

                  1  | 999 999   2 999

                  2  | 999   2 999 999
```

```
   3  | 999 999 999 999
```

the matrix is :

```
 u|v |   0   1   2   3
-----------------------
   0  |   1 999 999 999
   1  | 999 999   2 999
   2  | 999   2 999 999
   3  | 999 999 999 999
```

VErtex matrix:

```
 u|v |   0   1   2   3
-----------------------
   0  |   0   1   2   3
   1  |   0   1   2   3
   2  |   0   1   2   3
   3  |   0   1   2   3
```

the matrix is :

```
 u|v |   0   1   2   3
----------------------
   0  |   1 999 999 999
   1  | 999 999   2 999
   2  | 999   2 999 999
   3  | 999 999 999 999
```

VErtex matrix:

```
 u|v |   0   1   2   3
----------------------
   0  |   0   1   2   3
   1  |   0   1   2   3
   2  |   0   1   2   3
   3  |   0   1   2   3
```

the matrix is :

```
 u|v |   0   1   2   3
----------------------
   0  |   1 999 999 999
   1  | 999 999   2 999
   2  | 999   2 999 999
   3  | 999 999 999 999
```

VErtex matrix:

u|v	0	1	2	3
0	0	1	2	3
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3

the matrix is :

u|v	0	1	2	3
0	1	999	999	999
1	999	999	2	999
2	999	2	999	999
3	999	999	999	999

VErtex matrix:

u|v	0	1	2	3

```
-----------------------
  0  |   0   1   2   3
  1  |   0   1   2   3
  2  |   0   1   2   3
  3  |   0   1   2   3
```

the final matrix is :

```
 u|v |   0   1   2   3
-----------------------
  0  |   1 999 999 999
  1  | 999 999   2 999
  2  | 999   2 999 999
  3  | 999 999 999 999
```

Vertex matrix:

```
 u|v |   0   1   2   3
-----------------------
  0  |   0   1   2   3
  1  |   0   1   2   3
  2  |   0   1   2   3
  3  |   0   1   2   3
```

```
Enter the method you want to implement :
 1. Dijkshtra algorithm   2.Warshall Floyd   0.Exit
code::0

Thank you


        Graph Creation [Undirected/Directed]...
                Type of Graph [0: UnDirected] := 1
                How many vertices [upto 10 vertices]?? 5

        ...

        Verices starts at 0 and terminates at 4
                Vertex ID of -1 terminates Input

        Enter Existing Edges in the Graph

                ---------------------------------------------------
                Edge#   uVertex#   vVertex#   Distance      Remark
                ---------------------------------------------------
                    10
1
1
                                                          Edge Created
                    22
2
33
                                                          Edge Created
```

```
                    33
4
4
                                                                Edge Created
                    44
4
4
                                                                Edge Created
                    55


6
6
                                                                Invalid Edge
                    6-1
-1
-1
                                                                Invalid Edge


Enter the method you want to implement :
 1. Dijkshtra algorithm   2.Warshall Floyd   0.Exit
code::1


enter the source vertex :3
                      v |     0     1     2     3     4
                   prev |    -1    -1    -1    -1     3
                   dist |   999   999   999     0     4
                visited |     0     0     0     1     0

                      v |     0     1     2     3     4
                   prev |    -1    -1    -1    -1     3
```

```
                dist |   999   999   999     0     4
             visited |     0     0     0     1     1

                   v |     0     1     2     3     4
                prev |    -1    -1    -1    -1     3
                dist |   999   999   999     0     4
             visited |     0     0     1     1     1

                   v |     0     1     2     3     4
                prev |    -1    -1    -1    -1     3
                dist |   999   999   999     0     4
             visited |     0     1     1     1     1

                   v |     0     1     2     3     4
                prev |    -1    -1    -1    -1     3
                dist |   999   999   999     0     4
             visited |     1     1     1     1     1


Distance from source to 0: 999
Distance from source to 1: 999
Distance from source to 2: 999
Distance from source to 4: 4

0.
   0

1.
   1
```

2.

2

3.

3

4.

3--> 4

Enter the method you want to implement :

1. Dijkshtra algorithm 2.Warshall Floyd 0.Exit

code::2

enter the source vertex :3

u|v	0	1	2	3	4
0	999	1	999	999	999
1	999	999	999	999	999
2	999	999	33	999	999
3	999	999	999	999	4
4	999	999	999	999	4

the matrix is :

u|v | 0 1 2 3 4

```
-------------------------

  0  | 999   1 999 999 999

  1  | 999 999 999 999 999

  2  | 999 999  33 999 999

  3  | 999 999 999 999   4

  4  | 999 999 999 999   4
```

VErtex matrix:

```
 u|v |   0   1   2   3   4

-------------------------

  0  |   0   1   2   3   4

  1  |   0   1   2   3   4

  2  |   0   1   2   3   4

  3  |   0   1   2   3   4

  4  |   0   1   2   3   4
```

the matrix is :

```
 u|v |   0   1   2   3   4

-------------------------

  0  | 999   1 999 999 999

  1  | 999 999 999 999 999

  2  | 999 999  33 999 999
```

```
  3  | 999 999 999 999   4
  4  | 999 999 999 999   4
```

VErtex matrix:

```
 u|v |   0   1   2   3   4
-------------------------
  0  |   0   1   2   3   4
  1  |   0   1   2   3   4
  2  |   0   1   2   3   4
  3  |   0   1   2   3   4
  4  |   0   1   2   3   4
```

the matrix is :

```
 u|v |   0   1   2   3   4
-------------------------
  0  | 999   1 999 999 999
  1  | 999 999 999 999 999
  2  | 999 999  33 999 999
  3  | 999 999 999 999   4
  4  | 999 999 999 999   4
```

VErtex matrix:

u|v	0	1	2	3	4
0	0	1	2	3	4
1	0	1	2	3	4
2	0	1	2	3	4
3	0	1	2	3	4
4	0	1	2	3	4

the matrix is :

u|v	0	1	2	3	4
0	999	1	999	999	999
1	999	999	999	999	999
2	999	999	33	999	999
3	999	999	999	999	4
4	999	999	999	999	4

VErtex matrix:

u|v	0	1	2	3	4

```
-------------------------

  0  |   0   1   2   3   4

  1  |   0   1   2   3   4

  2  |   0   1   2   3   4

  3  |   0   1   2   3   4

  4  |   0   1   2   3   4
```

the matrix is :

```
 u|v |   0   1   2   3   4

-------------------------

  0  | 999   1 999 999 999

  1  | 999 999 999 999 999

  2  | 999 999  33 999 999

  3  | 999 999 999 999   4

  4  | 999 999 999 999   4
```

VErtex matrix:

```
 u|v |   0   1   2   3   4

-------------------------

  0  |   0   1   2   3   4

  1  |   0   1   2   3   4

  2  |   0   1   2   3   4
```

```
  3  |   0   1   2   3   4
  4  |   0   1   2   3   4
```

the final matrix is :

```
 u|v |   0   1   2   3   4
---------------------------
  0  | 999   1 999 999 999
  1  | 999 999 999 999 999
  2  | 999 999  33 999 999
  3  | 999 999 999 999   4
  4  | 999 999 999 999   4
```

Vertex matrix:

```
 u|v |   0   1   2   3   4
---------------------------
  0  |   0   1   2   3   4
  1  |   0   1   2   3   4
  2  |   0   1   2   3   4
  3  |   0   1   2   3   4
  4  |   0   1   2   3   4
```

```
Enter the method you want to implement :

 1. Dijkshtra algorithm   2.Warshall Floyd   0.Exit

code::0


Thank you
```

==========

VIVA-VOICE

==========

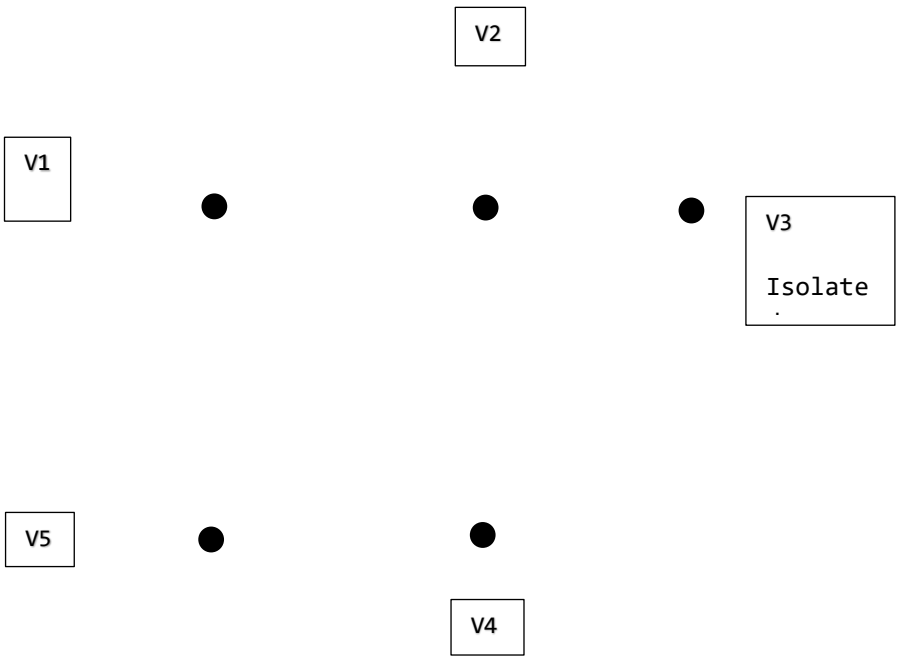
1. **What do you understand by an Eulerian walk?**

- *Eulerian Path:*
  Eulerian Path is a path in graph that visits every edge exactly once. Eulerian Circuit is an Eulerian Path which starts and ends on the same vertex.

2. **What is a null graph?**

- *Null Graph:*
  A null graph is defined as a graph which consists only the isolated vertices.

  Example: The graph shown in fig is a null graph, and the vertices are isolated vertices.



3. Explain topological sorting?

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

4. Distinguish between a open walk and closed walk in graph.

- *Open Walk :*

  A Walk is said to be OPEN if the first and the last vertices Are different I.e. the terminal vertices are different.

- *Closed Walk :*

  A Walk is said to be CLOSED if the first and the last vertices Are same I.e. the terminal vertices are same.

5. **Establish logical resemblance between Prim's method and Dijkstra's Approach**

Difference between Prim's and Dijkstra's Approach :

I. Dijkstra's algorithm finds the shortest path, but Prim's algorithm finds the MST.

II. Dijkstra's algorithm can work on both directed and undirected graphs, but Prim's algorithm only works on undirected graphs.

III. Prim's algorithm can handle negative edge weights, but Dijkstra's algorithm may fail to accurately compute distances if at least one negative edge weight exists

In practice, Dijkstra's algorithm is used when we want to save time and fuel traveling from one point to another. Prim's algorithm, on the other hand, is used when we want to minimize material costs in constructing roads that connect multiple points to each other.

==========

CONCLUSION

==========

In this experiment we have studied what is graph and its various operations. We have also learnt how to write the algorithms for operations of graphs. Sir has explained us in detail what is djashtra and warshell Floyd and its uses.