EXPERIMENT-10

Write a Program to implement Dijkstra's algorithm to compute the Shortest path through a graph.

Dijkstra's algorithm

Dijkstra's algorithm is used to find the shortest path from a single source vertex to all other vertices in a weighted graph, where all edge weights are non-negative.

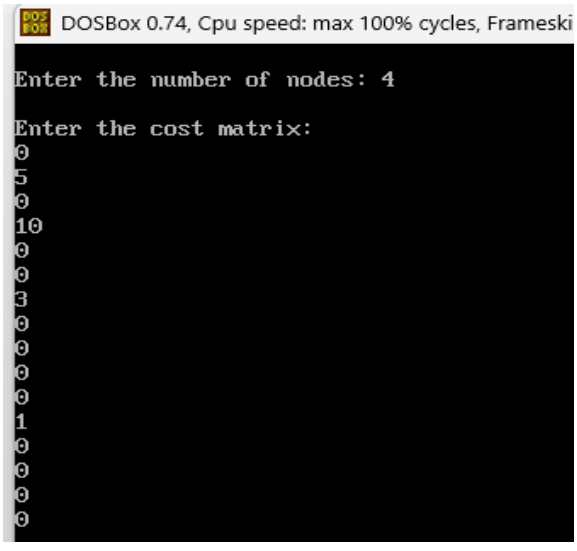
It ensures that for every vertex, the algorithm finds the minimum possible distance from the source.

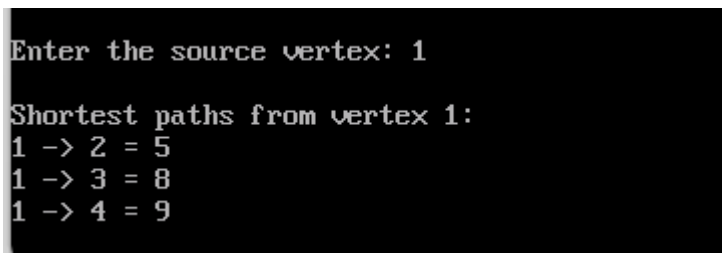
Program

```
#include <stdio.h>
#define INF 999
void dijkstra(int n, int v, int cost[10][10], int dist[])
{
   int i, u, count, w, flag[10], min;
   for (i = 1; i <= n; i++)
   {
      flag[i] = 0;
      dist[i] = cost[v][i];
   }
   flag[v] = 1;
   dist[v] = 0;
   count = 1;
   while (count < n)
   {
      min = INF;
      for (w = 1; w <= n; w++)
         if (dist[w] < min && !flag[w])
         {
               min = dist[w];
               u = w;
         }
      flag[u] = 1;
      count++;
      for (w = 1; w <= n; w++)
         if ((dist[u] + cost[u][w] < dist[w]) && !flag[w])
               dist[w] = dist[u] + cost[u][w];
   }}
```

```
void main()
{
   int n, v, i, j, cost[10][10], dist[10];
   clrscr();
   printf("\nEnter the number of nodes: ");
   scanf("%d", &n);
   printf("\nEnter the cost matrix:\n");
   for (i = 1; i <= n; i++)
   {
      for (j = 1; j <= n; j++)
      {
         scanf("%d", &cost[i][j]);
         if (cost[i][j] == 0 && i != j)
               cost[i][j] = INF;
      }
   }
   printf("\nEnter the source vertex: ");
   scanf("%d", &v);
   dijkstra(n, v, cost, dist);
   printf("\nShortest paths from vertex %d:\n", v);
   for (i = 1; i <= n; i++)
      if (i != v)
         printf("%d -> %d = %d\n", v, i, dist[i]);
   getch();
}
```

Output:





Don't Write in any ware from Here onwards

PROGRAM: Dijkstra's Algorithm (Shortest Path) Explaining step by step

```
#include <stdio.h>
#define INF 999
```

- **#include <stdio.h>** → Allows using standard input/output functions like printf() and scanf().
- **#define INF 999** → Defines a constant value representing **infinity** (a large number used when there's no direct path between nodes).

Dijkstra Function

```
void dijkstra(int n, int v, int cost[10][10], int dist[])
```

- Declares the **Dijkstra function**.
- **Parameters:**
  - n → total number of vertices (nodes)
  - v → source vertex (starting point)
  - cost[10][10] → cost adjacency matrix (graph)
  - dist[] → stores shortest distance from source v to each node

Declares variables:

```
int i, u, count, w, flag[10], min;
```

- i, w → loop counters
- u → vertex with minimum distance (chosen next)
- count → how many vertices have been finalized
- flag[10] → marks whether a vertex is already processed (1 = visited)
- min → temporary variable to hold smallest distance value found so far

Initialize all vertices:

```
for (i = 1; i <= n; i++)
{
   flag[i] = 0;
   dist[i] = cost[v][i];
}
```

- flag[i] = 0 → none of the nodes are visited yet.
- dist[i] = cost[v][i] → initial distance to each node is the **direct cost** from the source v.

```
flag[v] = 1;
dist[v] = 0;
count = 1;
```

- Mark the **source vertex** as visited (flag[v] = 1).
- Distance from source to itself is 0.
- Start with count = 1 because one vertex (the source) is already finalized.

MAIN LOOP — Repeat until all vertices are processed

```
while (count < n)
```

- Continue until we process all nodes.

```
min = INF;
for (w = 1; w <= n; w++)
  if (dist[w] < min && !flag[w])
  {
     min = dist[w];
     u = w;
  }
```

- Find the **unvisited vertex u** that has the **minimum distance** from the source.
- Initially, min = INF.
- Compare each node w:
- If it's not visited (!flag[w]) and its distance is smaller than min, then update u = w.

```
flag[u] = 1;
count++;
```

- Mark this node u as visited (its shortest path is now fixed).
- Increase count of visited nodes.

Update (Relaxation) Step

```
for (w = 1; w <= n; w++)
  if ((dist[u] + cost[u][w] < dist[w]) && !flag[w])
     dist[w] = dist[u] + cost[u][w];
```

- For every unvisited vertex w, check if the path from source → u → w is **shorter** than the current dist[w].
- If yes, update dist[w].

- This is the **core relaxation step** of Dijkstra's algorithm.

Function End

- After all vertices are processed, dist[] contains **the shortest distance from source** v **to every other vertex**.

Main Function

```
int main()

{

   int n, v, i, j, cost[10][10], dist[10];
```

Declares:

- n → number of nodes
- v → source vertex
- cost[][] → adjacency matrix
- dist[] → distance array to store results

```
printf("\nEnter the cost matrix:\n");
for (i = 1; i <= n; i++)
{
   for (j = 1; j <= n; j++)
   {
      scanf("%d", &cost[i][j]);
      if (cost[i][j] == 0 && i != j)
         cost[i][j] = INF;
   }
}
```

- Reads the adjacency matrix of graph.
- If cost[i][j] = 0 and i != j,
  it means **no direct edge**, so set cost[i][j] = INF.

Example:

```
0 5 0 10
0 0 3 0
0 0 0 1
0 0 0 0
```

Means:

- Edge 1→2 = 5
- Edge 1→4 = 10
- Edge 2→3 = 3
- Edge 3→4 = 1
- Others = no direct path.

```
printf("\nEnter the source vertex: ");
scanf("%d", &v);
```

- Reads which vertex to start from.

```
dijkstra(n, v, cost, dist);
```

- Calls the function to compute shortest paths.

```
printf("\nShortest paths from vertex %d:\n", v);
for (i = 1; i <= n; i++)
    if (i != v)
        printf("%d -> %d = %d\n", v, i, dist[i]);
```

- Displays the result:

For each vertex i, prints shortest distance from source v to i.