**Experiment No 11**

**Aim: To implement Dijkstra's routing algorithm**

**Theory**:

In computer networks, routing algorithms play a crucial role in determining the optimal path for data packets to travel from the source to the destination.

Routing algorithms are used to find the optimal path for data packets to travel from the source to the destination across a network. The efficiency and speed of data transmission depend heavily on the chosen routing algorithm.

Some common routing algorithms include:

1. **Distance Vector Routing Algorithm**:
2. **Link State Routing Algorithm**:
3. **Path Vector Routing Algorithm**:
4. **Flooding**:
5. **Hierarchical Routing**:

**Dijkstra's Algorithm:**

Dijkstra's algorithm is a famous algorithm used to find the shortest paths from a source node to all other nodes in a weighted graph. It is widely used in routing protocols and network pathfinding. Dijkstra's algorithm is associated with the **Link State Routing Algorithm**.

In the context of computer networks, the Link State Routing Algorithm uses Dijkstra's algorithm to compute the shortest path from the source node to all other nodes in the network.

**Theorem**: Given a graph *G=(V,E)* with non-negative edge weights, Dijkstra's algorithm finds the shortest path from a source vertex *s* to every other vertex in the graph.

**Algorithm**:

DIJKSTRA\_SHORTEST\_PATH(G, s, t):

dist[s] ← 0 π[s] ← NIL for each vertex v  V do if v ≠ s then dist[v] ← ∞ π[v] ← undefined end if

ENQUEUE(v, Q) // insert v to queue Q end for while Q is not empty do u ← vertex in Q having minimum dist[u] if u == t then break end if

DEQUEUE(u, Q) // Remove u from queue Q for each adjacent node v of u do val ← dist[u] + weight(u, v) if val < dist[v] then dist[v] ← val

π[v] ← u

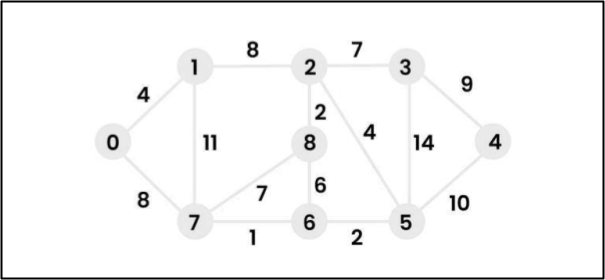
end if end for end while

**Explaination:**

1. **Initialization**:
   * Set the distance to the source node s to 0 and the distance to all other nodes to infinity.
   * Create a priority queue (min-heap) and insert all nodes with their current distances.
2. **Iteration**:
   * Extract the node *u* with the smallest distance from the priority queue.
   * For each neighbor *v* of *u*, calculate the tentative distance from the source *s* to *v* through u. If this distance is smaller than the current distance to *v*, update the distance and reinsert *v* into the priority queue.
3. **Completion**:
   * The algorithm terminates when the priority queue is empty.
   * The shortest path from the source to every other node is now known.

EXAMPLE:

**Input: src = 0, the graph is shown below.**



**Output: 0 4 12 19 21 11 9 8 14**

|  |  |
| --- | --- |
| **Explanation**: The distance from 0 to 1 = 4. |  |
| The minimum distance from 0 to 2 = 12. | 0->1->2 |
| The minimum distance from 0 to 3 = 19. | 0->1->2->3 |
| The minimum distance from 0 to 4 = 21. | 0->7->6->5->4 |
| The minimum distance from 0 to 5 = 11. | 0->7->6->5 |
| The minimum distance from 0 to 6 = 9. | 0->7->6 |
| The minimum distance from 0 to 7 = 8. | 0->7 |
| The minimum distance from 0 to 8 = 14. | 0->1->2->8 |

**Program Code:**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h> #define MAX\_V 100

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

{ int min = INT\_MAX, min\_index; for (int v = 0; v < V; v++)

{ if (sptSet[v] == 0 && dist[v] <= min)

{ min = dist[v]; min\_index = v;

} } return min\_index;

}

void dijkstra(int graph[MAX\_V][MAX\_V], int src, int V)

{ int dist[MAX\_V]; int sptSet[MAX\_V];

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

{ dist[i] = INT\_MAX; sptSet[i] = 0;

} dist[src] = 0; for (int count = 0; count < V - 1; count++)

{ int u = minDistance(dist, sptSet, V); sptSet[u] = 1;

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];

}

} }

printf("Vertex \t Distance from Source\n"); for (int i = 0; i < V; i++)

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

} }

int main()

{ int V; printf("Enter the number of vertices: "); scanf("%d", &V); int graph[MAX\_V][MAX\_V]; for (int i = 0; i < V; i++)

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

{ graph[i][j] = 0;

} }

printf("Enter the adjacency matrix (enter 0 if no edge):\n"); for (int i = 0; i < V; i++)

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

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

}

}

int src; printf("Enter the source vertex (0 to %d): ", V - 1); scanf("%d", &src); dijkstra(graph, src, V); return 0;

}

**Output:**

Enter the number of vertices: 5

Enter the adjacency matrix (enter 0 if no edge):

0 10 0 30 100

10 0 50 0 0

0 50 0 20 10

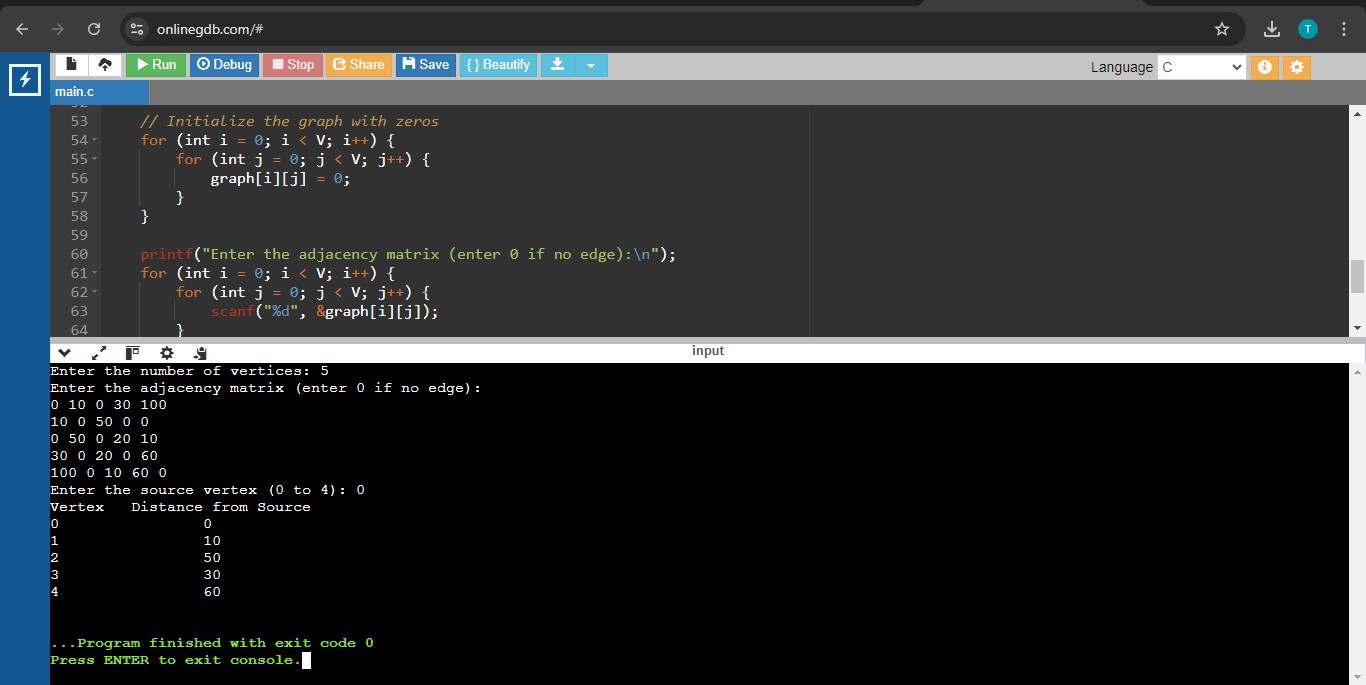
30 0 20 0 60

100 0 10 60 0

Enter the source vertex (0 to 4): 0

Vertex Distance from Source

1. 0
2. 10
3. 50
4. 30
5. 60



**Conclusion:**

We have successfully implemented Dijkshatra’s Routing Algorithm