Experiment No. : 3

Title: Implement Dijkstra`s Algorithm using Greedy approach

Batch:SY-IT(B3) **Roll No.: 16010423076** **Experiment No.: 3**

Aim: To implement and analyse time complexity Dijkstra's Algorithm to find Shortest path.

Explanation and Working of Dijkstra Algorithm:
Dijkstra's Algorithm is used to find the shortest path from a source node to all other nodes in a weighted graph. It works for graphs with non-negative edge weights and is commonly used in network routing and navigation systems.

Working :
- Start with the source node and assign it a distance of 0.
- Set the distance to all other nodes as infinity.
- Maintain a set of unvisited nodes.
- Pick the unvisited node with the smallest distance (initially the source).
- Update the distances of its neighbors by checking if a shorter path exists through the current node.
- Mark the current node as visited.
- Repeat the above steps until all nodes are visited or the shortest path to all nodes is found.
- Once complete, the shortest distances from the source to all nodes are determined.

Algorithm of Dijkstra Algorithm:
1. Initialize distances of all nodes as INFINITY, except the source node (distance = 0).
2. Create a priority queue (min-heap) to store nodes with their distances.
3. Add the source node to the queue with distance 0.
4. While the priority queue is not empty:
   a. Extract the node with the smallest distance (current node).
   b. For each neighbor of the current node:
      i. Calculate the new distance: distance[current] + edge weight.
      ii. If the new distance is smaller than the recorded distance:
         - Update the distance.
         - Add or update the neighbor in the priority queue.
   c. Mark the current node as visited.
5. Return the distances to all nodes.

Time complexity and derivation of Dijkstra Algorithm:
Time Complexity
Using a binary heap, the time complexity is O((V + E) log V).
In dense graphs, E ≈ V^2, so complexity approximates to O(V^2 log V).
In sparse graphs, E ≈ V, so complexity approximates to O(V log V).

Derivation
O(V * (pop vertex from min heap + no. of edges on each vertex * push in PQ))
O(V * (log(heapSize) + no. of edges * log(heapSize))

O(V * (log(heapSize) + V-1 * log(heapSize)) { one vertex can have V-1 edges }
O(V * (log(heapSize) * (V-1+1))
O(V * V * log(heapSize))

Now, at the worst case the heapSize will be equivalent to v^2 as if we consider pushing adjacent elements of a node, at the worst case each element will have V-1 nodes and they will be pushed onto the queue.

O(V * V * log(v^2))
O (v^2 * 2 log (V))
O (E * 2 log(V)) { E= v^2, total number of edges}
O (E * log(V)) Worst case Time Complexity of Dijkstra's Algorithm.

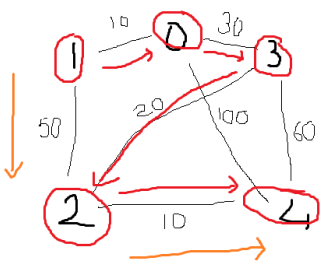
Program(s) of Dijkstra Algorithm:

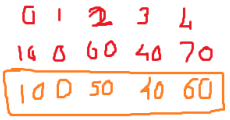
```
#include <stdio.h>
#include <limits.h>

//constants
#define INF INT_MAX
#define V 5 // Number of vertices

//find node with minimum distance
int findMinDistance(int dist[], int visited[]) {
   int min = INF, minIndex;
   for (int v = 0; v < V; v++) {
      if (!visited[v] && dist[v] <= min) {
         min = dist[v];
         minIndex = v;
      }
   }
   return minIndex;
}

//dijkstra's algorithm function
void dijkstra(int graph[V][V], int src) {
   int dist[V], visited[V];

   //initialization
   for (int i = 0; i < V; i++) {
      dist[i] = INF;
      visited[i] = 0;
   }
   dist[src] = 0;

   //process each node
   for (int count = 0; count < V - 1; count++) {
      int u = findMinDistance(dist, visited);
```

```
        visited[u] = 1;

        //update neighbour nodes
        for (int v = 0; v < V; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INF &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    //print results
    printf("Vertex\tDistance from Source\n");
    for (int i = 0; i < V; i++) {
        printf("%d\t%d\n", i, dist[i]);
    }
}

int main() {
    int graph[V][V] = {
        {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}
    };

    //calls graph with 0 as source node
    dijkstra(graph, 1);
    return 0;
}
```


Output(o) of Dijkstra Algorithm:

```
Output

Vertex  Distance from Source
0   10
1   0
2   50
3   40
4   60

=== Code Execution Successful ===
```





Conclusion: (Based on the observations):

From this experiment, I learned how Dijkstra's algorithm works to find the shortest paths in a weighted graph. I understood how to initialize distances, select the minimum distance node, and update the neighboring nodes step by step. The code helped me see how the algorithm prioritizes nodes and processes them efficiently.

Outcome: CO3 Implement Advance Programming algorithms with its application.

References:

1. **Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition**
2. **Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India**
3. **T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication**
4. **Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.**