



EXPERIMENT - 6

Algorithms Design and Analysis Lab

Aim

To implement Dijkstra's Algorithm and analyse its time complexity.

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 6

Aim:

To implement Dijkstra's Algorithm and analyse its time complexity.

Theory:

Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. Algorithm starts at the source vertex, s , it grows a tree, T , that ultimately spans all vertices reachable from S . Vertices are added to T in order of distance i.e., first S , then the vertex closest to S , then the next closest, and so on.

Following implementation assumes that graph G is represented by adjacency lists.

DIJKSTRA (G, w, s)

INITIALIZE SINGLE-SOURCE (G, s)

$S \leftarrow \{ \}$ // S will ultimately contains vertices of final shortest-path weights from s

Initialize priority queue Q i.e., $Q \leftarrow V[G]$

while priority queue Q is not empty do

$u \leftarrow \text{EXTRACT_MIN}(Q)$ // Pull out new vertex

$S \leftarrow S \cup \{u\}$ // Perform relaxation for each vertex v adjacent to u

 for each vertex v in $\text{Adj}[u]$ do

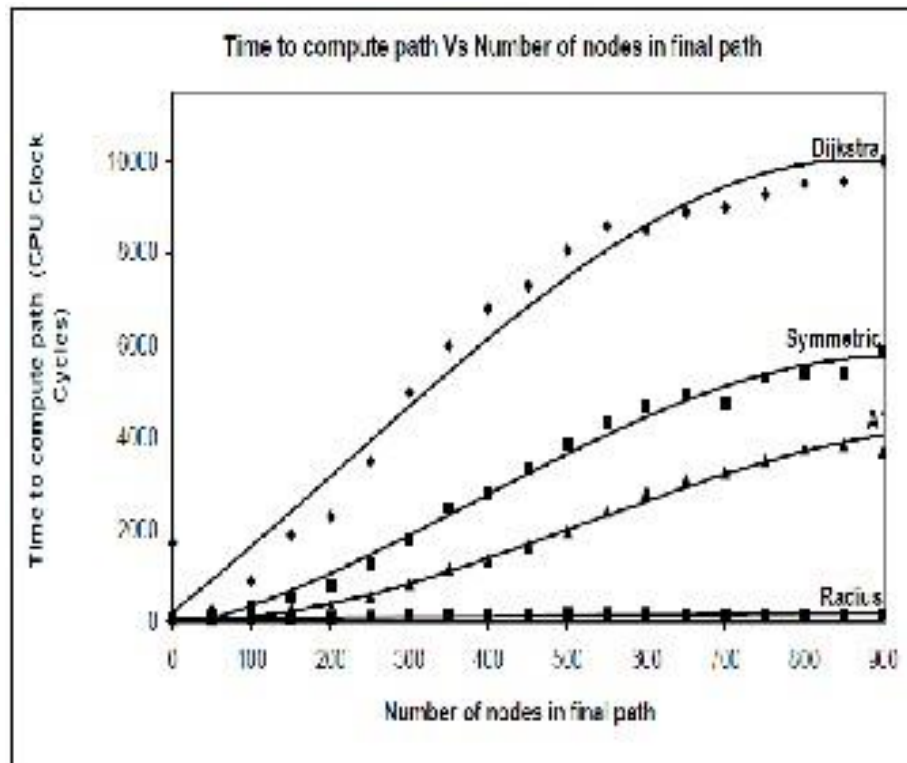
 Relax (u, v, w)

Implementation Steps:

- Enter a graph using adjacency list.
- Enter the cost of each edge.
- Enter the source vertex
- Find shortest distances for all vertices reachable from source.

Result and Analysis:

Like Prim's algorithm, Dijkstra's algorithm runs in $O(|E| \lg |V|)$ time.



Source Code:

```
#include <bits/stdc++.h>
using namespace std;

int miniDist(int distance[], bool Tset[]) // finding minimum distance
{
    int minimum = INT_MAX, ind;

    for (int k = 0; k < 6; k++)
    {
        if (Tset[k] == false && distance[k] <= minimum)
        {
            minimum = distance[k];
            ind = k;
        }
    }
}
```

```

    }
}
return ind;
}

void DijkstraAlgo(int graph[6][6], int src) // adjacency matrix
{
    int distance[6]; // // array to calculate the minimum distance for each node
    bool Tset[6];    // boolean array to mark visited and unvisited for each node

    for (int k = 0; k < 6; k++)
    {
        distance[k] = INT_MAX;
        Tset[k] = false;
    }

    distance[src] = 0; // Source vertex distance is set 0

    for (int k = 0; k < 6; k++)
    {
        int m = miniDist(distance, Tset);
        Tset[m] = true;
        for (int k = 0; k < 6; k++)
        {
            // updating the distance of neighbouring vertex
            if (!Tset[k] && graph[m][k] && distance[m] != INT_MAX && distance[m]
+ graph[m][k] < distance[k])
                distance[k] = distance[m] + graph[m][k];
        }
    }
    cout << "Vertex\t\tDistance from source vertex" << endl;
    for (int k = 0; k < 6; k++)
    {
        char str = 65 + k;
        cout << str << "\t\t" << distance[k] << endl;
    }
}

int main()
{
    int graph[6][6] = {
        {0, 1, 2, 0, 0, 0},
        {1, 0, 0, 5, 1, 0},
        {2, 0, 0, 2, 3, 0},
        {0, 5, 2, 0, 2, 2},
    }
}

```

```
{0, 1, 3, 2, 0, 1},
{0, 0, 0, 2, 1, 0}};

auto start = chrono::high_resolution_clock::now();
ios_base::sync_with_stdio(false);

DijkstraAlgo(graph, 0);

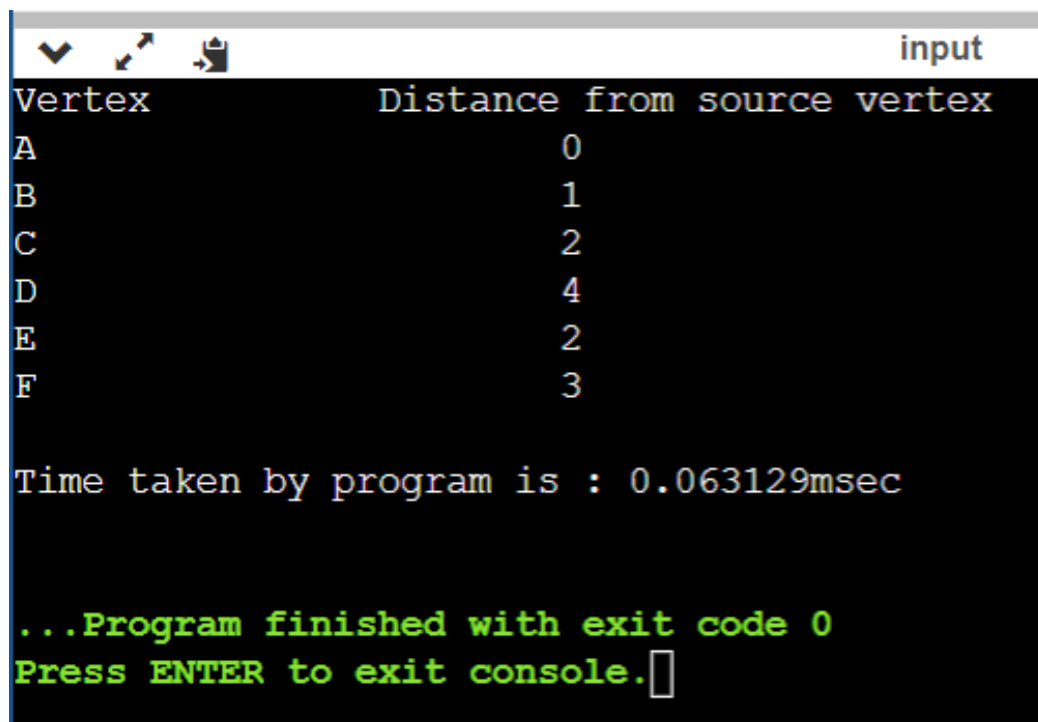
auto end = chrono::high_resolution_clock::now();
double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();

time_taken *= 1e-9 * 1000;

cout << "\nTime taken by program is : " << time_taken << setprecision(6);
cout << "msec" << endl;

return 0;
}
```

Output:



```
input
Vertex      Distance from source vertex
A           0
B           1
C           2
D           4
E           2
F           3

Time taken by program is : 0.063129msec

...Program finished with exit code 0
Press ENTER to exit console.
```

Viva Questions

1. What is the use of Dijkstra's algorithm?

Ans.

Dijkstra's procedure is used to solve the single-source shortest-paths method: for a given vertex called the source in a weighted linked graph, find the shortest path to all its other vertices. The single-source shortest-paths process asks for a family of paths, each leading from the source to various vertex in the graph, though some direction may have edges in common.

2. Give some applications of Dijkstra Algorithm.

Ans.

1. It is most widely used in finding shortest possible distance and show directions between 2 geographical locations such as in Google Maps.
2. This is also widely used in routing of data in networking and telecommunication domains for minimizing the delay occurred for transmission.
3. Wherever you encounter the need for shortest path solutions be it in robotics, transportation, embedded systems, factory or production plants to detect faults, etc this algorithm is used.

3. What is the most commonly used data structure for implementing Dijkstra's Algorithm?

Ans.

Minimum priority queue is the most commonly used data structure for implementing Dijkstra's Algorithm because the required operations to be performed in Dijkstra's Algorithm match with specialty of a minimum priority queue.

4. What is the time complexity of Dijkstra's algorithm?

Ans.

Time complexity of Dijkstra's algorithm is $O(N^2)$ because of the use of doubly nested for loops. It depends on how the table is manipulated.

5. The maximum number of times the decrease key operation performed in Dijkstra's algorithm will be equal to _____

Ans.

If the total number of edges in all adjacency list is E , then there will be a total of E number of iterations, hence there will be a total of at most E decrease key operations.