Lab Report-12

(Dijkstra's Algorithm)

CSE-2212 (Design and Analysis of Algorithms Lab)

Submitted By:

Name: Eyasir Ahamed
Exam Roll: 413
Class Roll: 15
Registration No: 202004017

Submitted To:

Sharad Hasan
Ex. Lecturer
Dept. of CSE
Sheikh Hasina University, Netrokona

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SHEIKH HASINA UNIVERSITY
NETROKONA, BANGLADESH

#12_Dijkstra’s Algorithm

Problem Definition: Given a weighted directed graph and a source vertex, Dijkstra's algorithm is used to find the shortest distance from the source vertex to all other vertices in the graph.

Formal Statement of Algorithm (Dijkstra's Algorithm):

- Initialize a priority queue pq to store pairs (distance, vertex) sorted by distance in non-decreasing order.
- Create a 1-indexed array distTo to store the shortest distances from the source vertex to all other vertices. Initialize all distances to infinity, except for the source vertex distance set to 0.
- Push the pair (0, source) onto the priority queue.
- While the priority queue is not empty, do the following:
  - Extract the pair (dist, prev) from the priority queue, where prev is the vertex with the shortest known distance from the source.
  - For each adjacent vertex next of prev, calculate the distance nextDist from the source through prev and update distTo[next] if it's shorter than the current distance.

    - Push the pair (distTo[next], next) onto the priority queue if distTo[next] was updated.
- After the algorithm terminates, the array distTo contains the shortest distances from the source vertex to all other vertices in the graph.

Complexity Analysis:

- Time Complexity: O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph. This complexity arises from the fact that each edge is processed once, and each edge addition to the priority queue takes logarithmic time.
- Space Complexity: O(V + E), where V is the number of vertices and E is the number of edges. The space complexity is dominated by the adjacency list representation of the graph and the priority queue.

Actual Code and Output

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

int main(){
    int n=5,m=6,source=1;
    vector<pair<int,int>> g[n+1]; // assuming 1 based indexing of graph
    // Constructing the graph
    g[1].push_back({2,2});
    g[1].push_back({4,1});
    g[2].push_back({1,2});
    g[2].push_back({5,5});
    g[2].push_back({3,4});
    g[3].push_back({2,4});
    g[3].push_back({4,3});
    g[3].push_back({5,1});
    g[4].push_back({1,1});
    g[4].push_back({3,3});
    g[5].push_back({2,5});
    g[5].push_back({3,1});
    // Dijkstra's algorithm begins from here
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;
    vector<int> distTo(n+1,INT_MAX);//1-indexed array for calculating shortest paths
    distTo[source] = 0;
    pq.push(make_pair(0,source));    // (dist,source)
    while( !pq.empty() ){
        int dist = pq.top().first;
        int prev = pq.top().second;
        pq.pop();
        for( auto it = g[prev].begin() ; it != g[prev].end() ; it++){
            int next = it->first;
            int nextDist = it->second;
            if( distTo[next] > distTo[prev] + nextDist){
                distTo[next] = distTo[prev] + nextDist;
                pq.push(make_pair(distTo[next], next));
            }
        }
    }
    cout << "The distances from source " << source << " are : \n";
    for(int i = 1 ; i<=n ; i++)    cout << distTo[i] << " ";
    cout << "\n";
    return 0;
}

```

```
The distances from source 1 are :
0 2 4 1 5
[Finished in 1.2s]
```