**Experimental Analysis on Dijkstra’s Algorithm to find the shortest path.**

**Introduction:**

In this report I am sharing my experience on the performance of Dijkstra’s Algorithm in finding the shortest path between two airports on different sized datasets.

**Importance:**

To begin with, Graphs do have lot of use cases in our day-to-day life. Finding the shortest paths between a source airport and destination airport is one such need to make our journey easier. So, finding such use cases is important and gives motivation to us.

**Theoretical Analysis:**

Dijkstra’s algorithm does have main steps, Initialization, and relaxation. Initialization is done only once, and Relaxation will be done only once for any node. So, in selecting that node, it uses a greedy approach.

As per the pseudocode of Dijkstra’s algorithm it is theoretically concluded that the algorithm takes O (|E| log |V|)time to create a shortest path tree.

So, we can infer that the time taken by algorithm will be depending on the sizes of nodes(|V|) and Edges(|E|)

**Practical Analysis:**

To test the algorithm, I have written a helper program to generate four random graphs (with increasing size). ***G1***: 25 nodes (i.e., airports) and 50 edges (i.e., direct flight links), ***G2***: 50 nodes and 200 edges, ***G3***: 100 nodes and 800 edges, ***G4***: 200 nodes and 3200 edges. Each edge is assigned a random integer weight between 300 (miles) to 2000 (miles). The nodes and edges are generated randomly. I used the ***adjacency list*** representation to implement/realize the graphs. In each of the graphs, I have randomly chosen the source s and destination t from the set of nodes of the graph.

I have considered 25 random sources and destinations for each of the above graphs and the time is taken to measure the performance.

Below is the table that represents the time taken by Dijkstra’s algorithm to find the shortest path between sources and destinations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num of Nodes, Num of Edges | 25, 50 | 50, 200 | 100, 800 | 200, 3200 |
| Time in Nano Seconds | 25900 | 235600 | 513500 | 757700 |

Based on the input sizes and average time taken by BFS algorithm, the above graph has been plotted.

**Analysis and Discussion:**

We can observe that the graph line is increasing as we increased the number of Nodes and number of Edges. So, we can say that if we increase the number of Nodes and Edges the time taken to compute the shortest path will be increased.

Calculations: The number of nodes is being doubled and number of edges is being quadrupled gradually in the given input sizes. That is why we can observe in the graph that line is increasing over time taken by the algorithm based on the performance of computer.

Output screenshot: The screenshot represents the output of BFS algorithm for given datasets.

Shape

Description automatically generated with medium confidence

***Hence, we can say that theoretical and actual performance are consistent***

**Conclusion:**

In a nutshell, we have implemented Dijkstra’s algorithm and then analyzed the performance of the algorithm on random input graph. We then inferred that theoretical/expected performance is consistent with actual performance.