****

**TASK**

**DEPARTMENT OF SOFTWARE ENGINEERING (SE)**

**SECTION: BS(SE)-Q**

**SUBMITTED TO:**

**Mam Noor ul Ain**

**SUBMITTED BY:**

**Muhammad Bin Awais 19I-0431**

**Salman**

**DATE OF SUBMISSION:**

**12-04-2022**

**Introduction**

**Graph-based structures are being increasingly used to model data and relations among data in a number of fields. Graph-based databases are becoming more popular as a means to better represent such data. Graph traversal is a key component in graph algorithms such as reachability and graph matching. Since the scale of data stored and queried in these databases is increasing, it is important to obtain high performing implementations of graph traversal that can efficiently utilize the processing power of modern processors. In this work, we present a scalable Breadth-First Search Traversal, Depth First traversal algorithm, MST using kruskal & Prims algorithm, Single source shortest path using dijskra and Bellmen ford algorithm for modern multi-socket, multi-core CPUs.**

* **Programming Language and Data Set**

**This project is done using python programming language. We have used python libraries like numpy,pandas etc for graph plotting and displaying. We didn’t use any built- in functions or libraries for kruskal,prims or any other algorithm used in the project.**

**We are working on a Bitcoin Alpha trust weighted signed network. This is who-trusts-whom network of people who trade using Bitcoin on a platform called** [**Bitcoin Alpha**](http://www.btcalpha.com/)**. Since Bitcoin users are anonymous, there is a need to maintain a record of users' reputation to prevent transactions with fraudulent and risky users. Members of Bitcoin Alpha rate other members on a scale of -10 (total distrust) to +10 (total trust) in steps of 1. This is the first explicit weighted signed directed network available for research.**

**Dijkstra Algorithm Trace :**

**Used priority queue data structure for calculating shortest path of given dataset.**

**1) Initialize distances of all vertices as infinite.**

**2) Create an empty priority\_queue pq. Every item of pq is a pair (weight, vertex). Weight (or distance) is used as the first item of a pair as the first item is by default used to compare two pairs.**

**3) Insert source vertex into pq and make its distance as 0.**

**4) While either pq doesn't become empty**

**a) Extract minimum distance vertex from pq. Let the extracted vertex be u.**

**b) Loop through all adjacent u and do the following for every vertex v.**

**5) Print distance array dist[] to print all shortest Paths.**

**Time Complexity: O(V2)**

**Auxiliary Space: O(V)**

**Bellman-Ford Algorithm Trace :**

**This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.Like other Dynamic Programming Problems, the algorithm calculates the shortest paths in a bottom-up manner. It first calculates the shortest distances which have at most one edge in the path. Then, it calculates the shortest paths with at-most 2 edges, and so on. After the i-th iteration of the outer loop, the shortest paths with at most i edges are calculated. There can be maximum |V| – 1 edges in any simple path, that is why the outer loop runs |v| – 1 times. The idea is, assuming that there is no negative weight cycle if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give the shortest path with at-most (i+1) edges.**

**Time Complexity Analysis :**

**O(V \* E), where V is the number of vertices in the graph and E is the number of edges in the graph**

**Auxiliary Space: O(E)**

**Kruskal Algorithm Trace :**

**Kruskal’s algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.**

**Time Complexity: O(ElogE) or O(ElogV), Sorting of edges takes O(ELogE) time. After sorting, we iterate through all edges and apply the find-union algorithm. Therefore, the overall time complexity is O(ElogE) or O(ElogV)**

**Auxiliary Space: O(V + E), where V is the number of vertices and E is the number of edges in the graph**

**Prim’s Algorithm Trace :**

The idea behind Prim’s algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning* Tree. And they must be connected with the minimum weight edge to make it a *Minimum* Spanning Tree.

**Time Complexity:** O(V2), If the input graph is represented using an adjacency list, then the time complexity of Prim’s algorithm can be reduced to O(E log V) with the help of a binary heap.

**Auxiliary Space:** O(V)

**Cycle Detection:**

**Check whether a given directed graph with 𝑛 vertices and 𝑚 edges contains a cycle.The following figure shows the classification of the edges encountered in DFS: Steps Use the recursive DFS implementation (pseudo-code shown in the below figure) Track if a node to be visited is already on the stack, if it’s there, it forms a back edge. Use parents array to obtain the directed cycle, if found.**

**Execution time of Algorithms :**

**Each algorithm complexity is analyzed and calculated along with the algorithm fuction**

**Machine specification :**

**All the test were run of XPS 15 corei i5.**