**REPORT OF RUN TIME ANALYSIS FOR MINIMUM SPANNING TREE USING PRIM’S AND KRUSKAL’S ALGORITHMS**

|  |  |
| --- | --- |
| **NAME:** | **UTA ID:** |
| MIRZA IMRAN BAIG | 1002086480 |
| VIVEK CHOWDARY GUNTUPALLI | 1002075917 |

**MINIMUM COST SPANNING TREE: -**

A minimum spanning tree or (MST) is a subgraph of an undirected graph such that the subgraph includes all nodes. The subgraph is connected, acyclic and has lowest total edge weight.

* Both prim’s and Kruskal’s Algorithms works with the undirected graphs
* Both are greedy algorithms that will produce optimal solutions

**Prims naive:- O(nm)**

**Pseudo Code Explanation: -**An MST must first be initialized with the randomly selected vertex. We now need to identify every edge that joins the tree from the previous stage with the fresh vertices. Choose the shortest edge from the discovered edges, then include it in the tree. Repeat the same procedure until the MST is produced. For searching shortest edge for every iteration of nodes(n) we iterate through at most (m) edges. So, the running time is O(nm)

Here “n” is denoted as nodes and “m” is denoted as edges

**Prims heap :- O(mlog(n))**

**Pseudo Code Explanation:-**Using a min priority queue, the time complexity for one call to EXTRACT-MIN(Q) is O(log n). The first while loop is executed n times in total. As a result, EXTRACT-MIN(Q) is referred to as n times. As a result, the complexity of EXTRACT-MIN(Q) is O. (n log(n)). For an undirected graph, the for loop is executed 2m times because the length of each adjacency list is 2m. By using the DECREASE KEY operation on the min heap, the time required to extract min is O(log n). As a result, the total execution time is O(2m log(n)) = O(m log(n)).

Here “n” is denoted as nodes and “m” is denoted as edges

**Kruskal’s naïve :- O(nm)**

**Pseudo Code Explanation :-**First, sort all of the edges from light to heavy. Now, add the edge with the least weight to the spanning tree. If the edge to be added creates a cycle, reject it. Sorting takes m log(m) time because we are sorting all the edges and for each edge we are using BFS traversing algorithm which takes O(n) time so the overall time complexity is O(nm)

Here “n” is denoted as nodes and “m” is denoted as edges

**Kruskal’s Union Find :- O((n+m)logn)**

**Pseudo Code Explanation : -**First, sort all of the edges from light to heavy. Now, add the edge with the least weight to the spanning tree. If the edge to be added creates a cycle, reject it. Sorting takes m log(m) time and we are using Union Find data structure to find if current node creates a cycle which takes at most n log(n) time so the time complexity is (m log(m) + n log(n)) but m is at most n2 so the time complexity is O(2.m log (n)+n log(n)) => O(m+n log(n))

Here “n” is denoted as nodes and “m” is denoted as edges

**OBSERVATION: -**

Though prims naïve and Kruskal’s naïve may have the same time complexity but the execution time for prims naïve is better than Kruskal’s naïve according to the experiment’s that we have performed and we found out that Kruskal’s heap performed better than all other algorithm’s however the execution time may be different for other machines.

The data that is used in these experiments is generated randomly and for each implementation though the data points are generated randomly but all the algorithms are executed using same data

**The result of the experiments is given below: -**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MINIMUM SPANNING TREE RUNNING TIME(S)** | | | | |
| Vertices, Edges | Prims naive | Prims heap | Kruskal’s naive | Kruskal’s Union Find |
| 100-1000 | 0.025 | 0.000 | 0.232 | 0.0 |
| 100-4000 | 0.018 | 0.002 | 1.090 | 0.001 |
| 200-5000 | 0.121 | 0.004 | 4.221 | 0.002 |
| 200-10000 | 0.132 | 0.01 | 9.296 | 0.008 |
| 300-5000 | 0.420 | 0.004 | 9.991 | 0.002 |
| 300-10000 | 0.443 | 0.016 | 20.08 | 0.006 |
| 400-10000 | 1.030 | 0.012 | 35.37 | 0.005 |
| 400-15000 | 1.062 | 0.021 | 56.952 | 0.013 |
| 500-10000 | 2.003 | 0.014 | 55.085 | 0.008 |
| 500-15000 | 2.043 | 0.024 | 83.152 | 0.008 |
| 500-25000 | 2.304 | 0.047 | 150.65 | 0.04 |
| 800-25000 | 8.683 | 0.043 | 386.58 | 0.020 |
| 800-40000 | 9.429 | 0.095 | 593.84 | 0.031 |
| 1000-20000 | 18.908 | 0.049 | 518.119 | 0.012 |
| 2000-20000 | 139.859 | 0.052 | 2056.548 | 0.018 |

* VIVEK CHOWDARY GUNTUPALLI (1002075917) have worked on prims naïve and prims heap implementations.
* MIRZA IMRAN BAIG (1002086480) have worked on Kruskal’s naïve and Kruskal’s Union Find implementations.