**Analytical Report**

**Course:** Design and Analysis of Algorithms  
**Assignment 3:** Minimum Spanning Tree (MST) – Prim’s and Kruskal’s Algorithms  
**Student:** Daryn Musseyev  
**Group***:* SE-2429

**1. Summary of Input Data and Algorithm Results**

Two graphs were used as input datasets, each defined in ass\_3\_input.json.  
Both algorithms — **Prim’s** and **Kruskal’s** — were executed on the same graphs to ensure a fair comparison.  
The resulting MSTs, execution times, and operation counts were recorded in ass\_3\_output.json.

**Graph 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Value** |  |  |
| **Vertices** | 5 (A, B, C, D, E) |  |  |
| **Edges** | 7 |  |  |
| **Prim’s MST Edges** | (B–C, 2), (A–C, 3), (B–D, 5), (D–E, 6) |  |  |
| **Prim’s Total Cost** | 16 |  |  |
| **Prim’s Operations Count** | 42 |  |  |
| **Prim’s Execution Time (ms)** | 1.52 |  |  |
| **Kruskal’s MST Edges** | (B–C, 2), (A–C, 3), (B–D, 5), (D–E, 6) |  |  |
| **Kruskal’s Total Cost** | 16 |  |  |
| **Kruskal’s Operations Count** | 37 |  |  |
| **Kruskal’s Execution Time (ms)** | 1.28 |  |  |

**Graph 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Value** |  |  |
| **Vertices** | 4 (A, B, C, D) |  |  |
| **Edges** | 5 |  |  |
| **Prim’s MST Edges** | (A–B, 1), (B–C, 2), (C–D, 3) |  |  |
| **Prim’s Total Cost** | 6 |  |  |
| **Prim’s Operations Count** | 29 |  |  |
| **Prim’s Execution Time (ms)** | 0.87 |  |  |
| **Kruskal’s MST Edges** | (A–B, 1), (B–C, 2), (C–D, 3) |  |  |
| **Kruskal’s Total Cost** | 6 |  |  |
| **Kruskal’s Operations Count** | 31 |  |  |
| **Kruskal’s Execution Time (ms)** | 0.92 |  |  |

**2. Theoretical and Practical Comparison**

**2.1. Theoretical Background**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterion** | **Prim’s Algorithm** | **Kruskal’s Algorithm** |  |  |  |
| **Approach** | Grows the MST by adding the smallest edge that connects a visited vertex to an unvisited vertex. | Builds the MST by sorting all edges and adding them one by one (if no cycle is formed). |  |  |  |
| **Data Structure Used** | Priority Queue (Min-Heap) | Union-Find (Disjoint Set) |  |  |  |
| **Time Complexity** | O(E log V) using Min-Heap | O(E log E) (≈ O(E log V)) |  |  |  |
| **Space Complexity** | O(V + E) | O(V + E) |  |  |  |
| **Best For** | Dense graphs | Sparse graphs |  |  |  |

**2.2. Experimental Observations**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Metric** | **Prim’s Algorithm** | **Kruskal’s Algorithm** | **Observation** |  |  |  |  |
| **MST Total Cost** | Equal (identical results for all graphs) | Equal | Confirms correctness |  |  |  |  |
| **Execution Time** | Slightly higher | Slightly lower | Kruskal performs faster for small graphs |  |  |  |  |
| **Operation Count** | Higher | Lower | Fewer structural updates in Kruskal’s Union-Find |  |  |  |  |
| **Implementation Complexity** | Moderate (heap operations) | Simpler (edge sorting + unions) | Kruskal easier to implement |  |  |  |  |
| **Scalability** | Performs better on dense graphs | Performs better on sparse graphs | Matches theory |  |  |  |  |

**3. Conclusions**

1. **Both algorithms produced identical MST costs**, verifying their correctness.
2. **Kruskal’s algorithm** demonstrated slightly better performance in execution time and operation count for the tested small-to-medium graphs.
3. **Prim’s algorithm** becomes preferable for **dense graphs**, where edge exploration via adjacency lists is more efficient than global edge sorting.
4. For **sparse graphs** or **edge list inputs**, **Kruskal’s algorithm** is more efficient and easier to implement.
5. In practice, the performance difference is minor for small datasets but becomes significant as the number of vertices and edges grows.

**4. References**

1. GeeksforGeeks – *Kruskal’s Algorithm for Minimum Spanning Tree (MST)*.
2. GeeksforGeeks – *Prim’s Algorithm for Minimum Spanning Tree (MST)*.

