**Report.**  
**Assignment 3.**

Sadvokassova Aliya

SE – 2405

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

Three graph datasets were analyzed to determine the Minimum Spanning Tree (MST) using **Prim’s** and **Kruskal’s** algorithms.  
 Each graph represents a connected weighted network where vertices are nodes and edges represent links with given weights (costs).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Graph ID** | **Vertices** | **Edges** | **Algorithm** | **Total Cost** | **Operations Count** | **Execution Time (ms)** |
| 1 | 5 | 7 | Prim | 16 | 28 | 0.7318 |
|  |  |  | Kruskal | 16 | 7 | 1.5092 |
| 2 | 12 | 15 | Prim | 31 | 165 | 0.0261 |
|  |  |  | Kruskal | 31 | 15 | 0.0273 |
| 3 | 25 | 26 | Prim | 121 | 624 | 0.0670 |
|  |  |  | Kruskal | 121 | 26 | 0.0644 |

**Interpretation:**

* For all three graphs, **Prim’s** and **Kruskal’s** algorithms produced identical MST total costs, confirming **correctness** of implementation.
* **Kruskal’s algorithm** consistently required fewer operations but had similar or slightly faster execution time only for larger graphs.
* **Prim’s algorithm** showed more operations but remained efficient for dense graphs with many vertices.
* As the graph size increases, operation count differences between the algorithms become more pronounced.

**2. Detailed MST Results (from output.json)**

**Graph 1**

* Vertices: 5  Edges: 7

Prim’s Algorithm MST Edges:

* A – C (3)
* B – C (2)
* B – D (5)
* D – E (6)  
   Total Cost: 16  Operations: 28  Execution Time: 0.7318 ms

Kruskal’s Algorithm MST Edges:

* B – C (2)
* A – C (3)
* B – D (5)
* D – E (6)  
   Total Cost: 16  Operations: 7  Execution Time: 1.5092 ms

**Graph 2**

* Vertices: 12  Edges: 15

Prim’s Algorithm MST Edges:

* A – B (2)
* A – C (3)
* C – D (1)
* D – F (5)
* F – H (4)
* B – E (5)
* E – G (2)
* G – I (3)
* I – K (2)
* K – L (3)
* J – L (1)  
   Total Cost: 31  Operations: 165  Execution Time: 0.0261 ms

Kruskal’s Algorithm MST Edges:

* C – D (1)
* J – L (1)
* A – B (2)
* E – G (2)
* I – K (2)
* A – C (3)
* G – I (3)
* K – L (3)
* F – H (4)
* D – F (5)
* B – E (5)  
   Total Cost: 31  Operations: 15  Execution Time: 0.0273 ms

**Graph 3**

* Vertices: 25  Edges: 26

Prim’s Algorithm MST Edges:  
 A – C (3), A – B (5), B – D (4), D – F (2), F – H (3), H – J (5), J – L (4),  
 C – E (6), E – G (7), L – N (7), N – P (3), P – R (4), R – T (6),  
 T – V (4), V – X (3), G – I (8), I – K (9), K – M (6), M – O (5),  
 O – Q (8), Q – S (5), S – U (7), U – W (2), W – Y (5)  
 Total Cost: 121  Operations: 624  Execution Time: 0.0670 ms

Kruskal’s Algorithm MST Edges:  
 D – F (2), U – W (2), A – C (3), F – H (3), N – P (3), V – X (3),  
 B – D (4), J – L (4), P – R (4), T – V (4), A – B (5), H – J (5),  
 M – O (5), Q – S (5), W – Y (5), C – E (6), K – M (6), R – T (6),  
 E – G (7), L – N (7), S – U (7), G – I (8), O – Q (8), I – K (9)  
 Total Cost: 121  Operations: 26  Execution Time: 0.0644 ms

**3. Comparison Between Prim’s and Kruskal’s Algorithms**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Prim’s Algorithm** | **Kruskal’s Algorithm** |
| **Approach** | Expands from a starting vertex, adding the smallest edge connecting to an unvisited vertex. | Sorts all edges and adds the smallest ones while avoiding cycles. |
| **Main Data Structures** | Visited set and edge list (can use priority queue). | Sorted edge list and Disjoint Set (Union-Find). |
| **Time Complexity** | O(V²) with adjacency list; O(E log V) with a heap. | O(E log E) due to sorting and union–find operations. |
| **Efficiency in Tests** | Higher operation counts (28, 165, 624). Slightly slower on larger graphs. | Fewer operations (7, 15, 26). More scalable. |
| **Memory Usage** | Depends on vertex representation. | Depends on edge representation. |
| **Best For** | Dense graphs or adjacency list representation. | Sparse graphs or edge list representation. |
| **Implementation Complexity** | Moderate. | Simple and clear. |

**Interpretation:**

* **Kruskal’s** algorithm generally outperforms **Prim’s** in terms of operation count for sparse and moderately sized graphs.
* **Prim’s** algorithm is better suited for dense graphs or when an adjacency matrix/heap is available.
* In all tests, both algorithms achieved identical MST total costs, proving correctness.

**4. Conclusions**

1. Both algorithms produced identical MST costs for all graphs, confirming their **correctness**.
2. **Kruskal’s algorithm** was consistently more efficient in terms of **operations count**, making it preferable for larger or sparser networks.
3. **Prim’s algorithm** can perform better on **dense graphs**, especially when implemented with a priority queue for edge selection.
4. Practical recommendations:  
    a. Use **Kruskal** for sparse or edge-list graphs.  
    b. Use **Prim** for dense or adjacency-based graphs.
5. Both algorithms remain key tools in network optimization, transportation planning, and cost-efficient connectivity design.

**4. References**

[Difference between Prim's and Kruskal's algorithm for MST - GeeksforGeeks](https://www.geeksforgeeks.org/dsa/difference-between-prims-and-kruskals-algorithm-for-mst/)[Kruskal's and Prim's Algorithms for Minimum Spanning Trees | Abdul Wahab Junaid](https://awjunaid.com/algorithm/kruskals-and-prims-algorithms-for-minimum-spanning-trees/)[Prim’s Algorithm for Minimum Spanning Tree (MST) - GeeksforGeeks](https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/)

[Kruskal’s Minimum Spanning Tree (MST) Algorithm - GeeksforGeeks](https://www.geeksforgeeks.org/dsa/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/)