**Temirbulatov Yeraly and Kadirbay Bibarys**

**SE-2412**

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

**Input Data Summary:**

* **Graph 1**: Composed of 5 nodes (A, B, C, D, E) and 7 edges, with a density of approximately 0.7 (relatively dense graph).
* **Graph 2**: Contains 4 nodes (A, B, C, D) and 5 edges, yielding a density of approximately 0.83 (very dense graph).

**Algorithm Results:**

**Graph 1:**

* **Kruskal's Algorithm**:
  + **Execution time**: 1.0 × 10⁻⁵ ms (0.00001 ms)
  + **Operations count**: 7
  + **Total MST cost**: 16
  + **MST edges**:
    - B-C (weight: 2)
    - A-C (weight: 3)
    - B-D (weight: 5)
    - D-E (weight: 6)
* **Prim's Algorithm**:
  + **Execution time**: 0.001 ms
  + **Operations count**: 7
  + **Total MST cost**: 16
  + **MST edges**:
    - A-C (weight: 3)
    - C-B (weight: 2)
    - B-D (weight: 5)
    - D-E (weight: 6)

**Graph 2:**

* **Kruskal's Algorithm**:
  + **Execution time**: 0.0 ms
  + **Operations count**: 5
  + **Total MST cost**: 6
  + **MST edges**:
    - A-B (weight: 1)
    - B-C (weight: 2)
    - C-D (weight: 3)
* **Prim's Algorithm**:
  + **Execution time**: 0.0 ms
  + **Operations count**: 5
  + **Total MST cost**: 6
  + **MST edges**:
    - A-B (weight: 1)
    - B-C (weight: 2)
    - C-D (weight: 3)

**2. Comparison of Prim's and Kruskal's Algorithms in Terms of Efficiency and Performance**

**Time Performance Evaluation:**

* Both algorithms demonstrated exceptional speed due to the small size of the graphs.
* **Kruskal's Algorithm** showed a non-zero execution time of 1.0 × 10⁻⁵ ms for Graph 1, while **Prim's Algorithm** showed a slightly higher execution time of 0.001 ms for Graph 1 and 0.0 ms for Graph 2.
* The low execution times indicate that both algorithms are highly efficient for small graphs like the ones in the test cases.

**Operational Efficiency:**

* The **operation count** corresponds to the number of edges in each graph:
  + Graph 1: 7 operations for 7 edges.
  + Graph 2: 5 operations for 5 edges.
* This confirms **optimal performance** for both algorithms, as they processed each edge exactly once.

**Result Validation:**

* Both **Kruskal's** and **Prim's** algorithms found **identical MSTs** with the same total cost:
  + **Graph 1**: Total MST cost = 16
  + **Graph 2**: Total MST cost = 6
* The **MST edge compositions** were equivalent, with only the order of edge addition differing between the two algorithms.

**3. Algorithm Selection: Recommendations Based on Experimental Findings**

**Observations from the Experiments:**

For small, dense graphs (such as the test cases):

* **Kruskal's** and **Prim's algorithms yield comparable performance**.
* **Execution time** is minimal and nearly identical for both algorithms, making the choice between them dependent on other factors.

**Advantages of Kruskal's Algorithm:**

* **Simpler to implement** due to reliance on edge sorting and the Union-Find structure.
* **Stable performance**, not affected by the starting node.
* **Potential for parallelization**, particularly for edge sorting.
* **Predictable operation count**, which is always equal to the number of edges.

**Advantages of Prim's Algorithm:**

* **Efficient for dense graphs**, as demonstrated in the test cases.
* **Incremental construction** of the MST starting from a chosen node.
* **Better asymptotic performance** on dense graphs when implemented with a Fibonacci heap.
* **Memory efficiency**, requiring O(V) space versus O(E) for Kruskal’s.

**Use Cases:**

**When to Use Kruskal's Algorithm**:

* When a **simple, reliable implementation** is required.
* When the graph is represented as an edge list.
* When **parallel processing** is needed.
* For **moderate-sized graphs**.

**When to Use Prim's Algorithm**:

* When dealing with **dense graphs**.
* When the **starting node** matters, or a specific node is needed to grow the MST.
* When an **adjacency matrix** is used for graph representation.
* When an efficient **Fibonacci heap** implementation is available.

**Key Considerations for Algorithm Selection:**

1. **Graph Size**: The performance difference is negligible for small graphs.
2. **Graph Density**: Prim's algorithm may be more efficient for dense graphs, while Kruskal’s works better for sparse graphs.
3. **Data Structure**: Kruskal’s is better with edge list representations, while Prim’s works better with adjacency matrices.
4. **Memory Requirements**: Kruskal requires O(E) space, Prim requires O(V) space.
5. **Implementation Complexity**: Kruskal’s is easier to implement and debug.

**4. Final Conclusions**

Based on the conducted experiments:

* Both **Kruskal's** and **Prim's** algorithms are highly efficient for small, dense graphs.
* **Identical results** were achieved in terms of MST cost and edge composition.
* **Minimal execution time** and **optimal operation counts** were observed for both algorithms.