**Analytical Report: Minimum Spanning Tree Optimization of a Transportation Network**

This report presents an analysis of Prim's and Kruskal's algorithms, implemented to determine the minimum construction cost for connecting city districts. The objective is to evaluate the correctness, operational efficiency, and execution time of both algorithms on various graph sizes to provide a comprehensive comparison.

---

## 1. Summary of Input Data and Algorithm Results

### 1.1 Test Dataset Profile

The analysis was conducted on a generated dataset comprising **30 weighted undirected graphs**. The graphs were automatically sized into four distinct categories (Small, Medium, Large, and Extra Large), with vertices (V) ranging from 11 up to 2,455. The dataset was specifically engineered to model a **sparse network topology**, where the number of edges (E) is significantly less than V(V−1)/2. This configuration is typical for urban infrastructure planning.

| Category | Avg. Vertices (V) | Avg. Edges (E) | Avg. Density Ratio (E/Emax) |
|---|---|---|---|
| Small (V ≤ 50) | 19.4 | 27.4 | 17.0% |
| Medium (V ≤ 350) | 179.3 | 356.6 | 2.4% |
| Large (V ≤ 1100) | 746 | 1862.2 | 0.7% |
| XL (V > 1100) | 2454.6 | 2944.2 | 0.1% |

Экспортировать в Таблицы

### 1.2 Validation and Cost Analysis

Both implemented algorithms successfully validated the fundamental properties of the Minimum Spanning Tree (MST):

- **Cost Congruence:** The **Total MST Cost** (Prim_Cost and Kruskal_Cost) was **identical** across all 30 test cases, confirming the high fidelity and correctness of both implementations. This fulfills the requirement that the MST cost must be reproducible regardless of the algorithm.
- **Structural Integrity:** Both algorithms consistently returned exactly **V−1 edges** for all connected graphs, verifying the acyclic and spanning nature of the resulting trees.

---

## 2. Comparison between Prim's and Kruskal's Algorithms (Efficiency and Performance)

### 2.1 Theoretical Time Complexity

The theoretical complexity provides the basis for expected performance under idealized conditions:

| Algorithm | Theoretical Time Complexity | Dominant Factor | Preferred Density |
|---|---|---|---|
| **Prim's (Binary Heap)** | O(ElogV) | Priority Queue operations (Extraction and Decrease-Key). | **Dense** (E≈V2) |
| **Kruskal's** | O(ElogE) | Initial sorting of all edges. | **Sparse**(E≪V2) |

Экспортировать в Таблицы

### 2.2 Practical Performance Analysis (Sparse Graphs)

The results demonstrate a clear practical efficiency gap between the two methods, primarily driven by the **sparse** nature of the test data.

#### A. Time Efficiency (Execution Time)

| Category | Prim Time (Avg. ms) | Kruskal Time (Avg. ms) | Speedup Ratio (P/K) | Time Advantage |
|---|---|---|---|---|
| Medium (V ≈ 179) | 0.4382 | 0.3694 | 1.19 | **Kruskal's** |
| Large (V ≈ 746) | 1.6617 | 1.2302 | **1.35** | **Kruskal's** |
| XL (V ≈ 2455) | 1.9881 | 2.0304 | 0.98 | **Parity** |

Экспортировать в Таблицы

**Time Conclusion:** For graphs up to V≈1000 (Large category), **Kruskal's Algorithm was measurably faster** (up to 35% faster). The execution time parity observed in the XL category suggests that the initial sorting overhead of Kruskal's becomes comparable to the cumulative Priority Queue overhead of Prim's at very large scales, even in sparse conditions.

#### B. Operational Efficiency (Key Actions)

The operational count (key algorithmic actions like queue access, comparisons, and DSU functions) confirms the practical difference in efficiency independent of system-level timing fluctuations.

| Category | Prim Operations (Avg.) | Kruskal Operations (Avg.) | Operational Ratio (K/P) | Operational Advantage |
|---|---|---|---|---|
| Medium | 2071.1 | 1006.6 | **0.49** | **Kruskal's** |
| Large | 10293.0 | 5390.0 | **0.52** | **Kruskal's** |
| XL | 18898.0 | 9406.0 | **0.50** | **Kruskal's** |

Экспортировать в Таблицы

**Operations Conclusion: Kruskal's Algorithm required approximately half the number of key operations** across all major size categories. This is the most critical finding, demonstrating the extremely high efficiency of the **Disjoint Set Union (DSU)** structure. While Prim's repeatedly accesses the Priority Queue and updates keys (costly operations), Kruskal's main operations are the initial sort and the near-constant-time efficiency ($O(\alpha(V))$) of the Union-Find operations.

---

### 3. Conclusions

#### 3.1 Preference by Graph Density and Representation

| Factor | Kruskal's Preference | Prim's Preference |
|---|---|---|
| **Graph Density** | **Sparse** (low edge count) | **Dense** (high edge count) |
| **Edge Representation** | **Edge List** (natural fit) | **Adjacency List** (required for neighbor traversal) |
| **Implementation Complexity** | Simpler (Relies on standard sorting and DSU) | More Complex (Requires robust Priority Queue management) |

Экспортировать в Таблицы

#### 3.2 Final Conclusion for the City Network

For the optimization of a city transportation network, modeled as a **sparse graph**, the following conclusion is drawn:

1. **Kruskal's Algorithm is the preferred choice** for this specific application. The practical results confirm its theoretical advantage in sparse conditions, offering a significant speedup (up to 35%) and requiring half the operational resources compared to Prim's Algorithm.
2. The primary efficiency gain in Kruskal's is not the initial sorting, but the performance of the **DSU data structure**, which efficiently prevents cycle

formation in nearly constant time, leading to overall faster convergence to the MST.

---

**4. References**


1. Assigned JSON input file (ass_3_input.json).
2. Generated CSV performance summary (performance_summary.csv).
3. Generated JSON output file (assignment_3_results.json).
4. Assignment Requirements Document.