**Analytical Report: Optimization of City Transportation Network  
Report Prepared By: Madiyar Kenzhebayev SE-2407   
For: Design and Analysis of Algorithms   
Assignment №3**

**Executive Summary**

This report analyzes the implementation and performance of Prim's and Kruskal's algorithms for finding Minimum Spanning Trees in city transportation networks. Both algorithms were successfully implemented and tested on multiple graph configurations, demonstrating identical minimum costs while revealing different performance characteristics based on graph density and size.

**1. Implementation Overview**

**Technical Architecture**

The solution employs a modular Java architecture with separate packages for algorithms, data models, and utilities. Key components include:

* **Graph Model**: Represents city districts as vertices and potential roads as weighted edges
* **Algorithm Interface**: Unified interface for MST algorithms enabling easy comparison
* **Performance Tracking**: Integrated operation counting and execution time measurement
* **JSON Processing**: Flexible input/output handling using Jackson library

**Algorithm Implementations**

**Prim's Algorithm**

* **Approach**: Vertex-growing strategy using priority queue
* **Data Structures**: Priority queue, adjacency lists, visited set
* **Complexity**: O(E log V) with binary heap
* **Key Features**: Greedy selection of minimum-weight edges connecting to current MST

**Kruskal's Algorithm**

* **Approach**: Edge-sorting strategy with union-find
* **Data Structures**: Union-Find (Disjoint Set Union) with path compression
* **Complexity**: O(E log E) for sorting + O(E α(V)) for union operations
* **Key Features**: Processes edges in ascending weight order

**2. Experimental Results**

**Test Case 1: 5 Districts, 7 Potential Roads**

**Input Statistics**

* Vertices: 5 (Districts A, B, C, D, E)
* Edges: 7 potential road connections
* Graph Density: Moderate (70% of possible edges)

**Algorithm Performance**

| Metric | Prim's Algorithm | Kruskal's Algorithm |
| --- | --- | --- |
| **Total Cost** | 16 | 16 |
| **Operations Count** | 42 | 37 |
| **Execution Time** | 1.52 ms | 1.28 ms |
| **MST Edges** | B-C(2), A-C(3), B-D(5), D-E(6) | B-C(2), A-C(3), B-D(5), D-E(6) |

**Analysis**: Kruskal's algorithm demonstrated superior performance with 12% fewer operations and 16% faster execution time for this moderately dense graph.

**Test Case 2: 4 Districts, 5 Potential Roads**

**Input Statistics**

* Vertices: 4 (Districts A, B, C, D)
* Edges: 5 potential road connections
* Graph Density: High (83% of possible edges)

**Algorithm Performance**

| Metric | Prim's Algorithm | Kruskal's Algorithm |
| --- | --- | --- |
| **Total Cost** | 6 | 6 |
| **Operations Count** | 29 | 31 |
| **Execution Time** | 0.87 ms | 0.92 ms |
| **MST Edges** | A-B(1), B-C(2), C-D(3) | A-B(1), B-C(2), C-D(3) |

**Analysis**: Prim's algorithm performed slightly better with 6.5% fewer operations and 5.4% faster execution for this denser graph.

**3. Algorithm Comparison**

**Performance Analysis**

**Time Complexity**

* **Prim's Algorithm**: O(E log V) with binary heap implementation
* **Kruskal's Algorithm**: O(E log E) dominated by edge sorting

**Space Complexity**

* **Prim's Algorithm**: O(V) for priority queue and visited set
* **Kruskal's Algorithm**: O(E) for edge storage and union-find structure

**Operational Efficiency**

| Graph Characteristic | Prim's Advantage | Kruskal's Advantage |
| --- | --- | --- |
| **Dense Graphs** | Better performance | - |
| **Sparse Graphs** | - | Superior efficiency |
| **Pre-sorted Edges** | - | Significant advantage |
| **Dynamic Graphs** | Easier updates | - |
| **Memory Constraints** | Lower memory usage | - |

**Key Findings**

1. **Cost Consistency**: Both algorithms consistently produced identical minimum spanning tree costs, validating implementation correctness.
2. **Performance Patterns**:
   * Kruskal excels with sparse graphs due to efficient edge processing
   * Prim performs better with dense graphs leveraging vertex-focused approach
   * The crossover point depends on the E/V ratio
3. **Implementation Complexity**:
   * Prim requires careful priority queue management
   * Kruskal benefits from efficient union-find implementation
   * Both algorithms are O(E log V) in practice for most cases

**4. Conclusions and Recommendations**

**Algorithm Selection Guidelines**

**Choose Prim's Algorithm When:**

* **Graph is dense** (many edges relative to vertices)
* **Using adjacency matrix** representation
* **Memory efficiency** is critical
* **Real-time updates** to the graph are required
* **Graph is connected** and relatively small

**Choose Kruskal's Algorithm When:**

* **Graph is sparse** (few edges relative to vertices)
* **Edges are pre-sorted** or easily sortable
* **Implementation simplicity** is prioritized
* **Dealing with disconnected components** (can find Minimum Spanning Forest)
* **Using adjacency list** representation

**Practical Recommendations for City Planning**

**For Urban Areas (Dense Networks)**

**Recommendation**: Prim's Algorithm

* Rationale: Urban transportation networks typically have high connectivity between districts
* Example: City centers with many interconnected roads

**For Suburban/Rural Areas (Sparse Networks)**

**Recommendation**: Kruskal's Algorithm

* Rationale: Suburban areas have fewer direct connections between districts
* Example: Residential areas with limited road infrastructure

**For Large-Scale Planning**

**Recommendation**: Hybrid Approach

* Use Kruskal for initial edge filtering on large datasets
* Apply Prim for final optimization on reduced edge sets
* Consider parallel processing for very large networks

**Performance Optimization Strategies**

1. **Data Structure Selection**:
   * Use Fibonacci heaps for Prim's algorithm in performance-critical applications
   * Implement union-find with path compression and union-by-rank for Kruskal
2. **Memory Management**:
   * For large graphs, consider edge streaming approaches
   * Use efficient data representations to reduce memory footprint
3. **Parallel Processing**:
   * Edge sorting in Kruskal's algorithm can be parallelized
   * Prim's algorithm offers limited parallelization opportunities

**5. Limitations and Future Work**

**Current Limitations**

* Memory-intensive for very large graphs (>10,000 nodes)
* Limited to static graph structures
* No support for dynamic weight updates

**Potential Enhancements**

1. **Incremental MST Updates**: Support for adding/removing edges without full recomputation
2. **Distributed Computing**: Parallel implementation for massive transportation networks
3. **Real-time Optimization**: Integration with live traffic data and dynamic cost adjustments
4. **Multi-objective Optimization**: Consider factors beyond construction cost (traffic flow, environmental impact)

**6. References**

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
2. Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6), 1389-1401.
3. Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1), 48-50.
4. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.

**Appendix: Implementation Verification**

**Correctness Validation**

* Both algorithms produced identical MST costs for all test cases
* MST properties verified (V-1 edges, no cycles, minimum total weight)
* Edge cases tested including single-node graphs and complete graphs

**Performance Validation**

* Operation counts consistent with theoretical complexity
* Execution times scale predictably with input size
* Memory usage within expected bounds