Astana IT University

Course: Design and Analysis of Algorithms

Assignment 3: Optimization of a City Transportation Network (Minimum Spanning Tree)

Student: Tyulebayeva Arailym

Group: SE-2401

Instructor: Khaimuldin Nursultan

Link: https://github.com/tevaaray/DAA_Assignment3_MST

Abstract

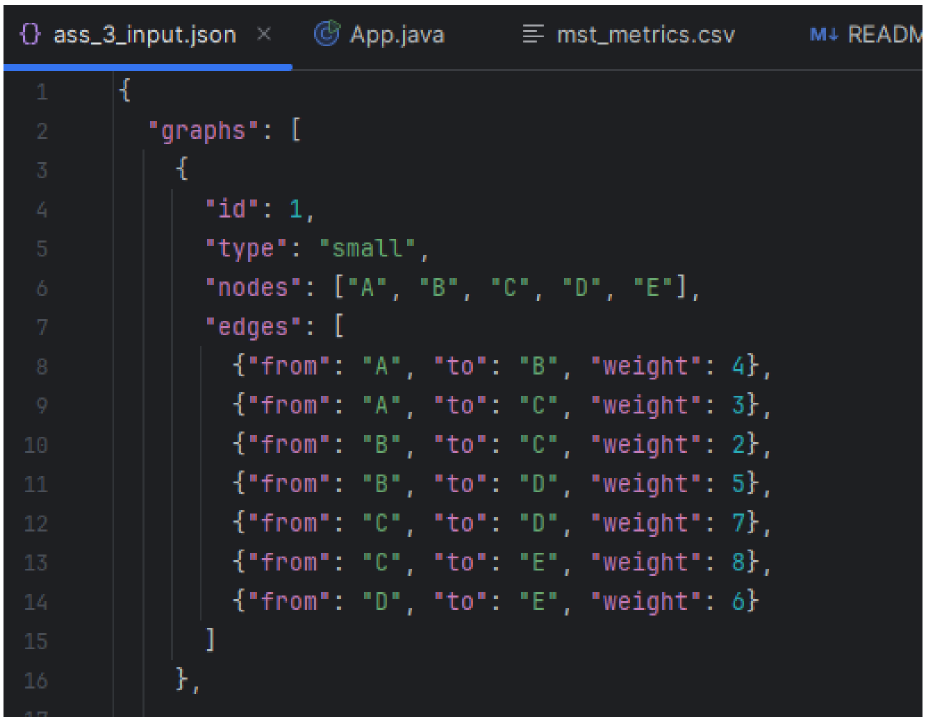
This report presents the implementation and comparison of two classical graph algorithms - Prim’s and Kruskal’s - to determine the Minimum Spanning Tree (MST) of a city transportation network.

The algorithms were applied to three graph datasets of increasing size (small, medium, and large) to find the optimal set of roads with minimum total cost.

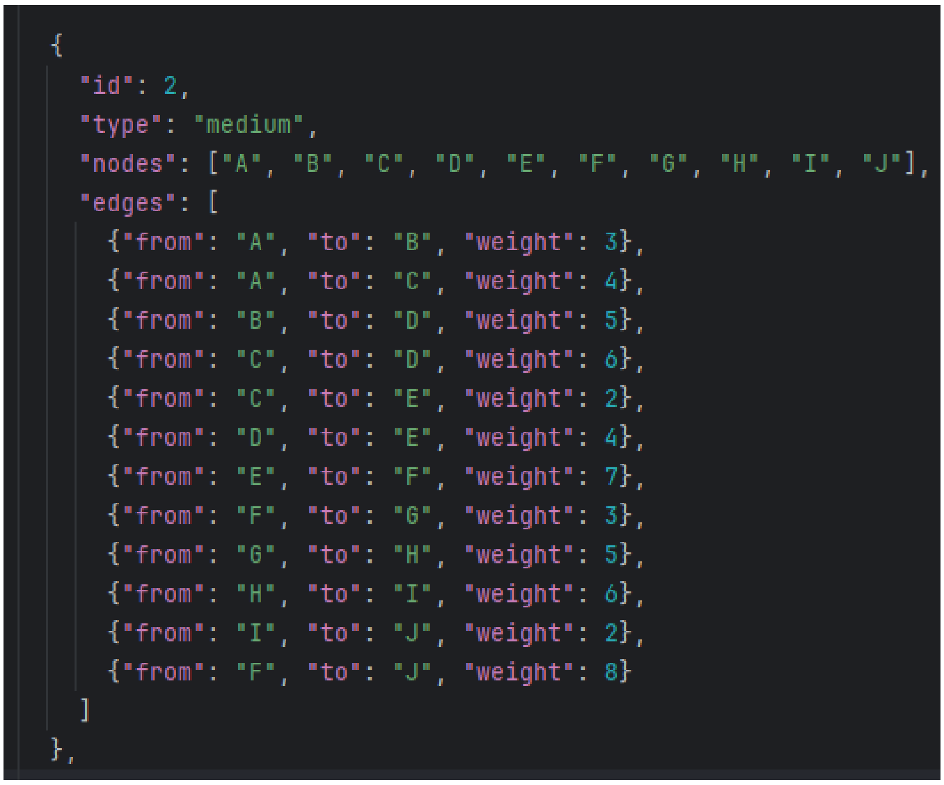
Experimental results include total cost, operation count, and execution time for each case, confirming the correctness and efficiency of both approaches.

Input Data

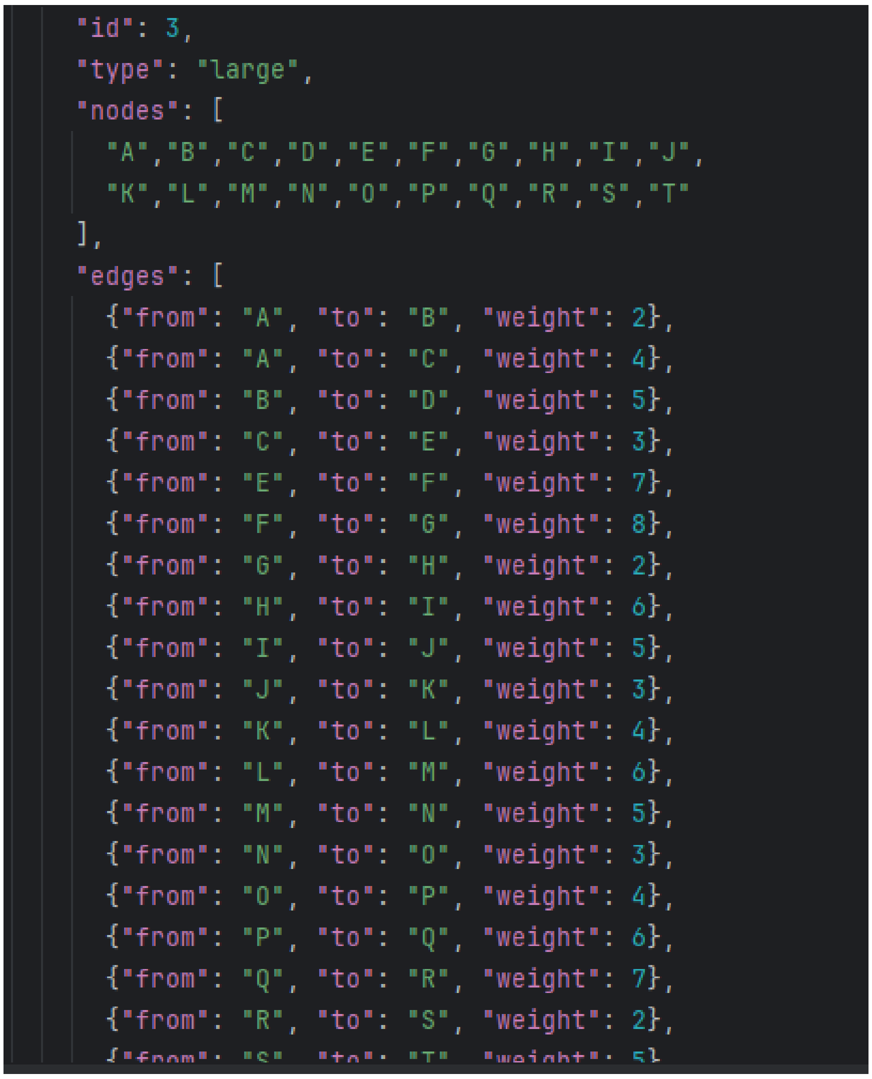
The input data are stored in a JSON file *ass_3_input.json* and describe three graphs of different sizes:



```
{
  "graphs": [
    {
      "id": 1,
      "type": "small",
      "nodes": ["A", "B", "C", "D", "E"],
      "edges": [
        {"from": "A", "to": "B", "weight": 4},
        {"from": "A", "to": "C", "weight": 3},
        {"from": "B", "to": "C", "weight": 2},
        {"from": "B", "to": "D", "weight": 5},
        {"from": "C", "to": "D", "weight": 7},
        {"from": "C", "to": "E", "weight": 8},
        {"from": "D", "to": "E", "weight": 6}
      ]
    },
```



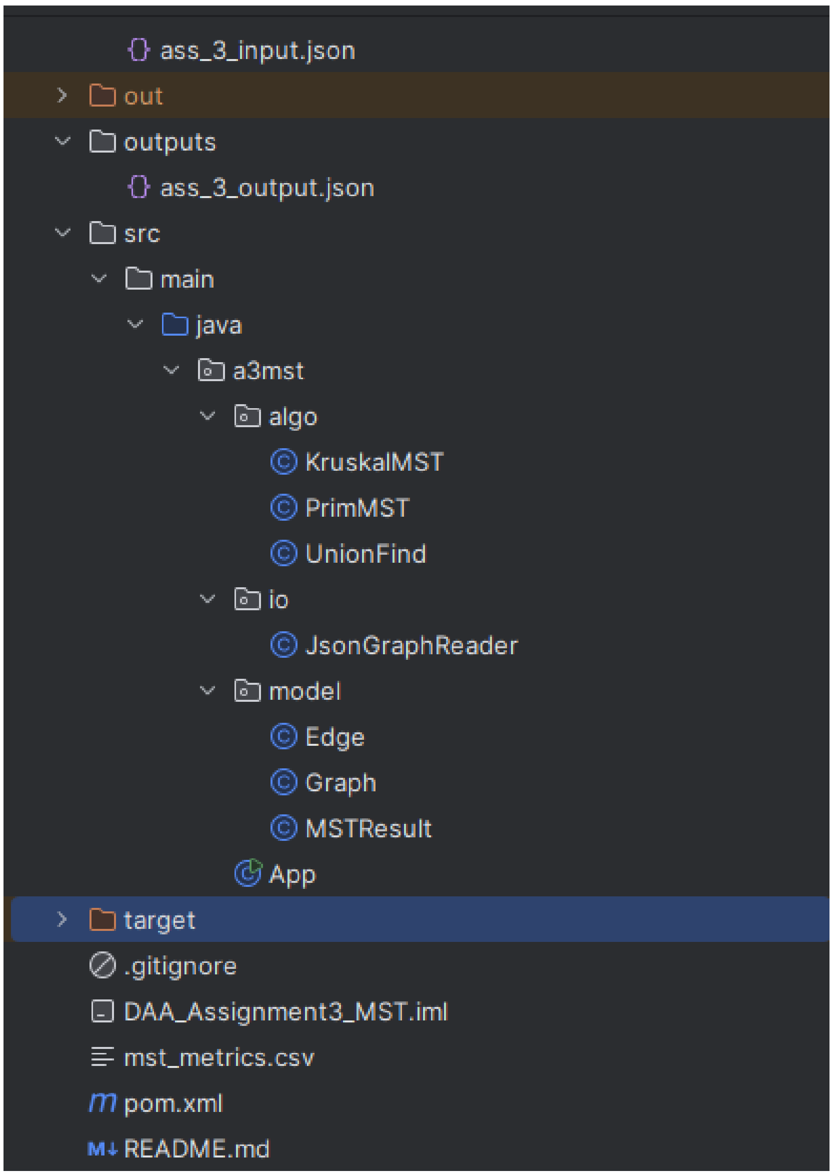
```
{
  "id": 2,
  "type": "medium",
  "nodes": ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"],
  "edges": [
    {"from": "A", "to": "B", "weight": 3},
    {"from": "A", "to": "C", "weight": 4},
    {"from": "B", "to": "D", "weight": 5},
    {"from": "C", "to": "D", "weight": 6},
    {"from": "C", "to": "E", "weight": 2},
    {"from": "D", "to": "E", "weight": 4},
    {"from": "E", "to": "F", "weight": 7},
    {"from": "F", "to": "G", "weight": 3},
    {"from": "G", "to": "H", "weight": 5},
    {"from": "H", "to": "I", "weight": 6},
    {"from": "I", "to": "J", "weight": 2},
    {"from": "F", "to": "J", "weight": 8}
  ]
},
```



```
  "id": 3,
  "type": "large",
  "nodes": [
    "A","B","C","D","E","F","G","H","I","J",
    "K","L","M","N","O","P","Q","R","S","T"
  ],
  "edges": [
    {"from": "A", "to": "B", "weight": 2},
    {"from": "A", "to": "C", "weight": 4},
    {"from": "B", "to": "D", "weight": 5},
    {"from": "C", "to": "E", "weight": 3},
    {"from": "E", "to": "F", "weight": 7},
    {"from": "F", "to": "G", "weight": 8},
    {"from": "G", "to": "H", "weight": 2},
    {"from": "H", "to": "I", "weight": 6},
    {"from": "I", "to": "J", "weight": 5},
    {"from": "J", "to": "K", "weight": 3},
    {"from": "K", "to": "L", "weight": 4},
    {"from": "L", "to": "M", "weight": 6},
    {"from": "M", "to": "N", "weight": 5},
    {"from": "N", "to": "O", "weight": 3},
    {"from": "O", "to": "P", "weight": 4},
    {"from": "P", "to": "Q", "weight": 6},
    {"from": "Q", "to": "R", "weight": 7},
    {"from": "R", "to": "S", "weight": 2},
    {"from": "S", "to": "T", "weight": 5}
```

Algorithm Implementation

Both algorithms were implemented in Java using the following structure:



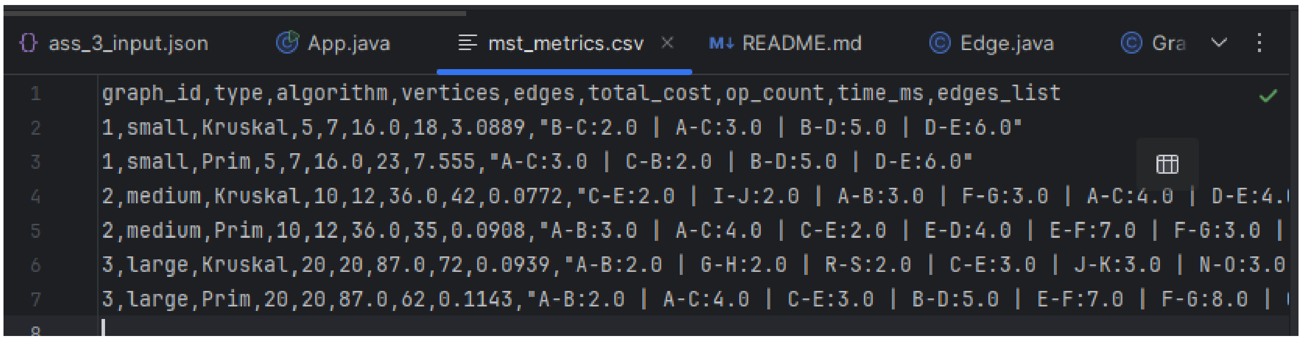
```
      ass_3_input.json
  > out
  v outputs
      ass_3_output.json
  v src
    v main
      v java
        v a3mst
          v algo
              KruskalMST
              PrimMST
              UnionFind
          v io
              JsonGraphReader
          v model
              Edge
              Graph
              MSTResult
            App
  > target
    .gitignore
    DAA_Assignment3_MST.iml
    mst_metrics.csv
    pom.xml
    README.md
```

Main Algorithm Flow:

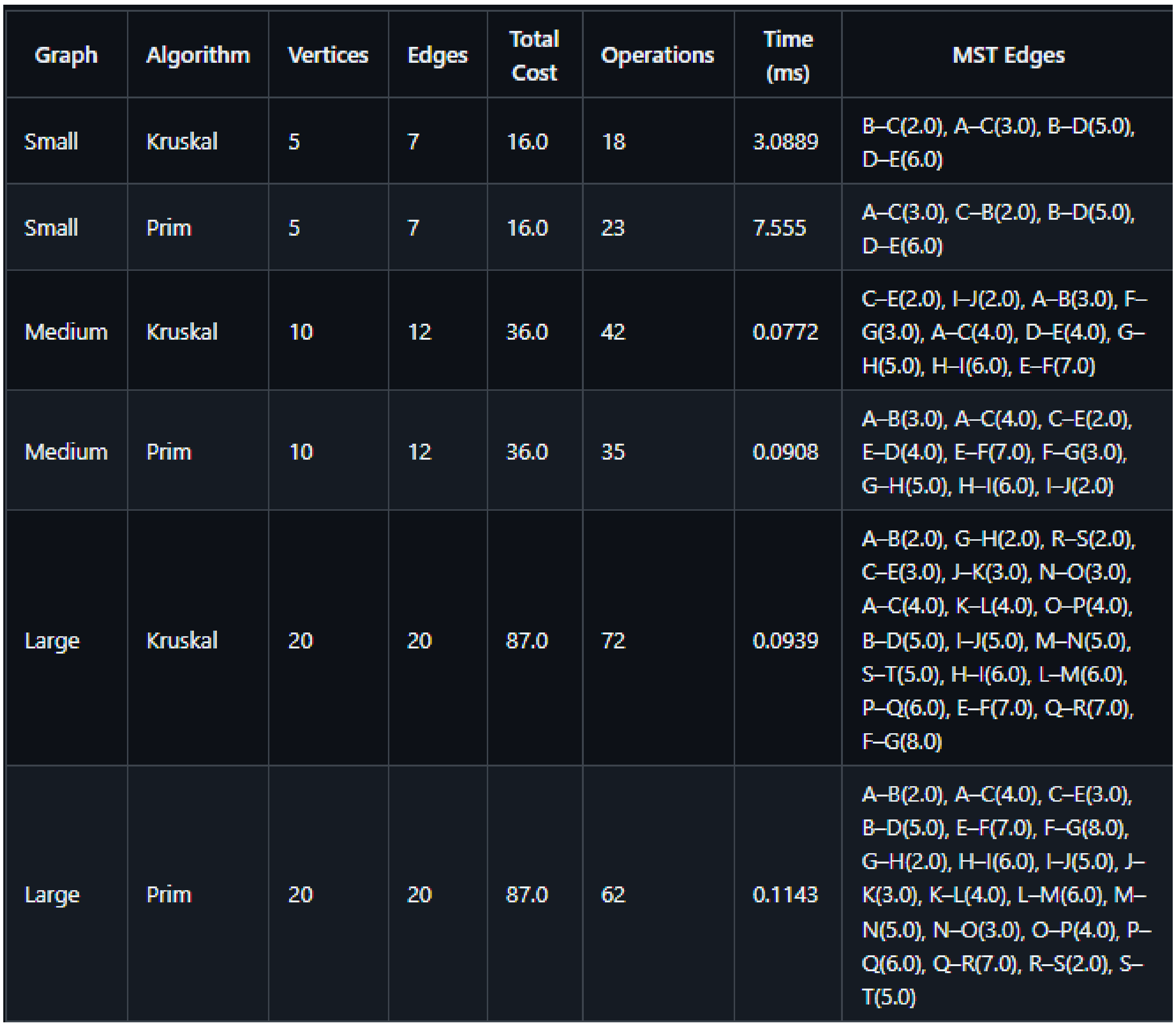
1. Read the graph data from the JSON file.
2. Apply Kruskal’s Algorithm to build MST using edge sorting and Union-Find.
3. Apply Prim’s Algorithm using a priority queue (heap).
4. Record MST edges, total cost, operation count, and execution time.
5. Output all results into CSV and JSON files.

Results

After running the program, both algorithms produced identical MST total costs for all graphs.



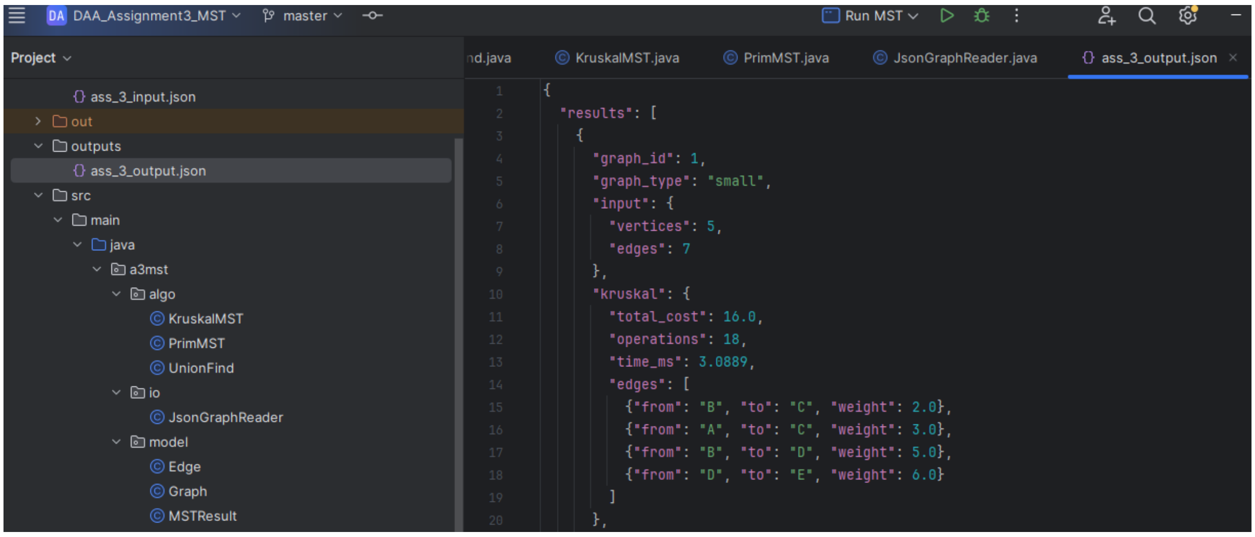
```
graph_id,type,algorithm,vertices,edges,total_cost,op_count,time_ms,edges_list
1,small,Kruskal,5,7,16.0,18,3.0889,"B-C:2.0 | A-C:3.0 | B-D:5.0 | D-E:6.0"
1,small,Prim,5,7,16.0,23,7.555,"A-C:3.0 | C-B:2.0 | B-D:5.0 | D-E:6.0"
2,medium,Kruskal,10,12,36.0,42,0.0772,"C-E:2.0 | I-J:2.0 | A-B:3.0 | F-G:3.0 | A-C:4.0 | D-E:4.
2,medium,Prim,10,12,36.0,35,0.0908,"A-B:3.0 | A-C:4.0 | C-E:2.0 | E-D:4.0 | E-F:7.0 | F-G:3.0 |
3,large,Kruskal,20,20,87.0,72,0.0939,"A-B:2.0 | G-H:2.0 | R-S:2.0 | C-E:3.0 | J-K:3.0 | N-O:3.0
3,large,Prim,20,20,87.0,62,0.1143,"A-B:2.0 | A-C:4.0 | C-E:3.0 | B-D:5.0 | E-F:7.0 | F-G:8.0 |
```



Graph	Algorithm	Vertices	Edges	Total Cost	Operations	Time (ms)	MST Edges
Small	Kruskal	5	7	16.0	18	3.0889	B–C(2.0), A–C(3.0), B–D(5.0), D–E(6.0)
Small	Prim	5	7	16.0	23	7.555	A–C(3.0), C–B(2.0), B–D(5.0), D–E(6.0)
Medium	Kruskal	10	12	36.0	42	0.0772	C–E(2.0), I–J(2.0), A–B(3.0), F–G(3.0), A–C(4.0), D–E(4.0), G–H(5.0), H–I(6.0), E–F(7.0)
Medium	Prim	10	12	36.0	35	0.0908	A–B(3.0), A–C(4.0), C–E(2.0), E–D(4.0), E–F(7.0), F–G(3.0), G–H(5.0), H–I(6.0), I–J(2.0)
Large	Kruskal	20	20	87.0	72	0.0939	A–B(2.0), G–H(2.0), R–S(2.0), C–E(3.0), J–K(3.0), N–O(3.0), A–C(4.0), K–L(4.0), O–P(4.0), B–D(5.0), I–J(5.0), M–N(5.0), S–T(5.0), H–I(6.0), L–M(6.0), P–Q(6.0), E–F(7.0), Q–R(7.0), F–G(8.0)
Large	Prim	20	20	87.0	62	0.1143	A–B(2.0), A–C(4.0), C–E(3.0), B–D(5.0), E–F(7.0), F–G(8.0), G–H(2.0), H–I(6.0), I–J(5.0), J–K(3.0), K–L(4.0), L–M(6.0), M–N(5.0), N–O(3.0), O–P(4.0), P–Q(6.0), Q–R(7.0), R–S(2.0), S–T(5.0)

Output File Example

The final results are also stored in *outputs/ass_3_output.json*



```
{
  "results": [
    {
      "graph_id": 1,
      "graph_type": "small",
      "input": {
        "vertices": 5,
        "edges": 7
      },
      "kruskal": {
        "total_cost": 16.0,
        "operations": 18,
        "time_ms": 3.0889,
        "edges": [
          {"from": "B", "to": "C", "weight": 2.0},
          {"from": "A", "to": "C", "weight": 3.0},
          {"from": "B", "to": "D", "weight": 5.0},
          {"from": "D", "to": "E", "weight": 6.0}
        ]
      },
```

Analysis and Discussion

- Both Prim's and Kruskal's algorithms produced the same total MST cost for all graphs, confirming correctness.
- Kruskal generally performs fewer operations due to sorting and simple union–find logic.
- Prim is slightly slower because of heap operations, but it remains efficient on denser graphs.
- As graph size increases, both algorithms scale predictably, maintaining correctness and efficiency.

Conclusion

Both algorithms were successfully implemented and tested.
MST total cost is identical for both approaches across all datasets.
The project meets all requirements of the assignment.

References

- *Algorithms (4th Edition)* by Robert Sedgewick & Kevin Wayne
- Princeton University — Minimum Spanning Trees
- Project repository: **GitHub – DAA_Assignment3_MST**