Ramazan Kozhabek

SE-2426

Assignment 3

Report on Prim’s and Kruskal’s algorithms

1. **A summary of input data and algorithm results (algorithm used, execution time, and operation count for each data case);**

   Graph 1 (Vertices: 5)

   - Prim: Total Cost = 132, Execution Time = 3.617 ms, Operations = 8

   - Kruskal: Total Cost = 132, Execution Time = 6.0984 ms, Operations = 6

   Graph 2 (Vertices: 10)

   - Prim: Total Cost = 291, Execution Time = 0.1288 ms, Operations = 27

   - Kruskal: Total Cost = 291, Execution Time = 0.06 ms, Operations = 17

   Graph 3 (Vertices: 15)

   - Prim: Total Cost = 349, Execution Time = 0.2317 ms, Operations = 63

   - Kruskal: Total Cost = 349, Execution Time = 0.1501 ms, Operations = 35

   Graph 4 (Vertices: 20)

   - Prim: Total Cost = 518, Execution Time = 0.2706 ms, Operations = 89

   - Kruskal: Total Cost = 518, Execution Time = 0.1733 ms, Operations = 51

   Graph 5 (Vertices: 25)

   - Prim: Total Cost = 583, Execution Time = 0.1881 ms, Operations = 91

   - Kruskal: Total Cost = 583, Execution Time = 0.1724 ms, Operations = 64

Graph 6 (Vertices: 30)

- Prim: Total Cost = 1115, Execution Time = 0.2036 ms, Operations = 102

- Kruskal: Total Cost = 1115, Execution Time = 0.1332 ms, Operations = 59

Graph 7 (Vertices: 55)

- Prim: Total Cost = 1642, Execution Time = 0.4158 ms, Operations = 221

- Kruskal: Total Cost = 1642, Execution Time = 1.4511 ms, Operations = 117

Graph 8 (Vertices: 80)

- Prim: Total Cost = 1752, Execution Time = 0.6938 ms, Operations = 353

- Kruskal: Total Cost = 1752, Execution Time = 0.6719 ms, Operations = 207

Graph 9 (Vertices: 110)

- Prim: Total Cost = 2174, Execution Time = 1.1063 ms, Operations = 497

- Kruskal: Total Cost = 2174, Execution Time = 1.0223 ms, Operations = 312

Graph 10 (Vertices: 140)

- Prim: Total Cost = 3544, Execution Time = 1.4716 ms, Operations = 583

- Kruskal: Total Cost = 3544, Execution Time = 1.4075 ms, Operations = 306

Graph 11 (Vertices: 170)

- Prim: Total Cost = 3743, Execution Time = 1.8487 ms, Operations = 781

- Kruskal: Total Cost = 3743, Execution Time = 1.5468 ms, Operations = 446

Graph 12 (Vertices: 200)

- Prim: Total Cost = 5300, Execution Time = 1.1402 ms, Operations = 715

- Kruskal: Total Cost = 5300, Execution Time = 1.1702 ms, Operations = 398

Graph 13 (Vertices: 230)

- Prim: Total Cost = 3960, Execution Time = 1.2392 ms, Operations = 1226

- Kruskal: Total Cost = 3960, Execution Time = 1.386 ms, Operations = 674

Graph 14 (Vertices: 260)

- Prim: Total Cost = 5226, Execution Time = 1.0747 ms, Operations = 1256

- Kruskal: Total Cost = 5226, Execution Time = 1.1419 ms, Operations = 701

Graph 15 (Vertices: 300)

- Prim: Total Cost = 6425, Execution Time = 1.0672 ms, Operations = 1427

- Kruskal: Total Cost = 6425, Execution Time = 1.2671 ms, Operations = 836

Graph 16 (Vertices: 320)

- Prim: Total Cost = 8545, Execution Time = 1.1157 ms, Operations = 1208

- Kruskal: Total Cost = 8545, Execution Time = 0.7975 ms, Operations = 672

Graph 17 (Vertices: 380)

- Prim: Total Cost = 9711, Execution Time = 1.1927 ms, Operations = 1577

- Kruskal: Total Cost = 9711, Execution Time = 1.1262 ms, Operations = 877

Graph 18 (Vertices: 440)

- Prim: Total Cost = 9132, Execution Time = 1.7544 ms, Operations = 2309

- Kruskal: Total Cost = 9132, Execution Time = 1.198 ms, Operations = 1211

Graph 19 (Vertices: 520)

- Prim: Total Cost = 14240, Execution Time = 1.4188 ms, Operations = 2070

- Kruskal: Total Cost = 14240, Execution Time = 1.2929 ms, Operations = 1090

Graph 20 (Vertices: 600)

- Prim: Total Cost = 13386, Execution Time = 2.0754 ms, Operations = 3123

- Kruskal: Total Cost = 13386, Execution Time = 1.7011 ms, Operations = 1644

Graph 21 (Vertices: 680)

- Prim: Total Cost = 17240, Execution Time = 2.8647 ms, Operations = 2848

- Kruskal: Total Cost = 17240, Execution Time = 2.3568 ms, Operations = 1520

Graph 22 (Vertices: 760)

- Prim: Total Cost = 22384, Execution Time = 2.0422 ms, Operations = 3092

- Kruskal: Total Cost = 22384, Execution Time = 1.7894 ms, Operations = 1556

Graph 23 (Vertices: 840)

- Prim: Total Cost = 18995, Execution Time = 5.0141 ms, Operations = 4061

- Kruskal: Total Cost = 18995, Execution Time = 2.7177 ms, Operations = 2175

Graph 24 (Vertices: 920)

- Prim: Total Cost = 19030, Execution Time = 4.3549 ms, Operations = 5003

- Kruskal: Total Cost = 19030, Execution Time = 2.0245 ms, Operations = 2708

Graph 25 (Vertices: 1000)

- Prim: Total Cost = 19770, Execution Time = 2.204 ms, Operations = 5670

- Kruskal: Total Cost = 19770, Execution Time = 2.0497 ms, Operations = 2947

Graph 26 (Vertices: 1100)

- Prim: Total Cost = 29740, Execution Time = 1.2282 ms, Operations = 4454

- Kruskal: Total Cost = 29740, Execution Time = 1.6018 ms, Operations = 2286

Graph 27 (Vertices: 1500)

- Prim: Total Cost = 37466, Execution Time = 1.8486 ms, Operations = 6643

- Kruskal: Total Cost = 37466, Execution Time = 3.1856 ms, Operations = 3424

Graph 28 (Vertices: 1900)

- Prim: Total Cost = 41328, Execution Time = 4.57 ms, Operations = 10163

- Kruskal: Total Cost = 41328, Execution Time = 4.4489 ms, Operations = 5393

2. **A comparison between Prim’s and Kruskal’s algorithms in terms of efficiency and performance (Theory and In Practice);**

Feature	Prim's Algorithm	Kruskal's Algorithm
Approach	Vertex-based, grows the MST one vertex at a time	Edge-based, adds edges in increasing order of weight
Data Structure	Priority queue (min-heap)	Union-Find data structure
Graph Representation	Adjacency matrix or adjacency list	Edge list
Initialization	Starts from an arbitrary vertex	Starts with all vertices as separate trees (forest)
Edge Selection	Chooses the minimum weight edge from the connected vertices	Chooses the minimum weight edge from all edges
Cycle Management	Not explicitly managed; grows connected component	Uses Union-Find to avoid cycles
Complexity	O(V^2) for adjacency matrix, O((E + V) log V) with a priority queue	O(E log E) or O(E log V), due to edge sorting

As we can see here in table from geeksforgeeks,both algorithms are used for same purpose but have different approach,data structure and complexity.Unlike Prim's algorithm,Kruskal's algorithm does not require starting from single vertex ;it selects edges globally based on weight.In theory it means Kruskal's algorithm should have fewer operations and less time.In Practice,we can see the same pattern:

9	4	Kruskal	20	518	0.1733	51
10	5	Prim	25	583	0.1881	91
11	5	Kruskal	25	583	0.1724	64
12	6	Prim	30	1115	0.2036	102
13	6	Kruskal	30	1115	0.1332	59
14	7	Prim	55	1642	0.4158	221
15	7	Kruskal	55	1642	1.4511	117
16	8	Prim	80	1752	0.6938	353
17	8	Kruskal	80	1752	0.6719	207
18	9	Prim	110	2174	1.1063	497
19	9	Kruskal	110	2174	1.0223	312
20	10	Prim	140	3544	1.4716	583
21	10	Kruskal	140	3544	1.4075	306

In most cases Kruskal’s algorithm is more effective.

But in large graphs situation is opposite:

52	26	Prim	1100	29740	1.2282	4454
53	26	Kruskal	1100	29740	1.6018	2286
54	27	Prim	1500	37466	1.8486	6643
55	27	Kruskal	1500	37466	3.1856	3424
56	28	Prim	1900	41328	4.57	10163
57	28	Kruskal	1900	41328	4.4489	5393

Even though Kruskal still takes fewer operations,it needs more time than Prim.

While Kruskal is efficient for sparse graphs, in large or dense graphs it may take **more time than Prim** due to the overhead of sorting a large number of edges and performing many union-find operations.

And one more thing:

42	21	Prim	680	17240	2.8647	2848
43	21	Kruskal	680	17240	2.3568	1520
44	22	Prim	760	22384	2.0422	3092
45	22	Kruskal	760	22384	1.7894	1556
46	23	Prim	840	18995	5.0141	4061
47	23	Kruskal	840	18995	2.7177	2175

The number of operations and execution time do not increase linearly because graph density, edge distribution, and algorithmic data structure behavior affect how many steps are actually performed and how costly each step is. Small differences in graph structure can produce significant variations in performance.

3. **Conclusions discussing which algorithm is preferable under different conditions (e.g., graph density, edge representation, implementation complexity etc.);**

Graph Density

- Sparse graphs (few edges relative to vertices): Kruskal’s algorithm is often faster and simpler to implement because it sorts a relatively small number of edges and avoids repeatedly checking all edges.
- Dense graphs (many edges): Prim’s algorithm is usually preferable since it only considers edges connecting the growing MST to remaining vertices, reducing unnecessary operations.

Edge Representation

- Adjacency matrix: Prim works efficiently because it can quickly find the smallest edge for each vertex.
- Edge list: Kruskal is more convenient, as it naturally sorts edges and applies union-find operations.

Implementation Complexity

- Prim requires a priority queue or heap to efficiently select edges, which adds some implementation complexity.

- Kruskal requires a disjoint-set (union-find) data structure, which is straightforward with path compression and union by rank.

Operation and Time Considerations

- Kruskal may perform fewer key operations, but sorting all edges can dominate execution time on very large or dense graphs.
- Prim may perform more operations in sparse graphs but is often faster on dense graphs due to localized edge selection.

Summary Recommendation

- Use Kruskal for sparse graphs or when you already have an edge list.
- Use Prim for dense graphs or when your graph is represented as an adjacency matrix.
- Both algorithms produce the same MST for connected graphs, so correctness is guaranteed; the choice mainly depends on graph characteristics and performance requirements.