Vikram Thirumaran

Prof. Thanh H. Nguyen
CS 315

January 19th, 2025

HW3

Q1: Basics

Q1.1: Minimum Spanning Trees

First, we construct an MST using Kruskal & Prim's Algorithm separately

Solution Using Kruskal's Algorithm

Step by step of **Kruskal's Algorithm**:

1. First, we sort all the edges in increasing order of weight.
2. Then initialize an empty MST(no edges).
3. Iterate through the sorted edges:
    - If an edge connects two different components (i.e., does not create a cycle), add it to the MST.
    - Use Union-Find (Disjoint Set) to efficiently check and merge components.
4. (Recursive step)Repeat until the MST has *n−1* edges, where *n* is the number of nodes.

Step 1: Sort All Edges by Weight

1. (J,K)(J, K)(J,K) – 2
2. (F,G)(F, G)(F,G) – 2
3. (F,J)(F, J)(F,J) – 3
4. (B,F)(B, F)(B,F) – 4
5. (C,F)(C, F)(C,F) – 4
6. (H,L)(H, L)(H,L) – 4
7. (E,F)(E, F)(E,F) – 5
8. (G,K)(G, K)(G,K) – 5
9. (F,G)(F, G)(F,G) – 5
10. (I,J)(I, J)(I,J) – 5
11. (D,G)(D, G)(D,G) – 6

12. (E,F)(E, F)(E,F) – 6
13. (A,B)(A, B)(A,B) – 9
14. (A,E)(A, E)(A,E) – 9
15. (C,D)(C, D)(C,D) – 9
16. (K,L)(K, L)(K,L) – 9
17. (B,C)(B, C)(B,C) – 10
18. (F,I)(F, I)(F,I) – 10
19. (G,H)(G, H)(G,H) – 8

Step 2: Add Edges to the MST

- Add (J,K)(J, K)(J,K) – 2
- Add (F,G)(F, G)(F,G) – 2
- Add (F,J)(F, J)(F,J) – 3
- Add (B,F)(B, F)(B,F) – 4
- Add (C,F)(C, F)(C,F) – 4
- Add (E,F)(E, F)(E,F) – 5
- Add (I,J)(I, J)(I,J) – 5
- Add (G,H)(G, H)(G,H) – 8
- Add (A,B)(A, B)(A,B) – 9

Final MST Using Kruskal's Algorithm + Edges in MST (in order of addition):
(J,K),(F,G),(F,J),(B,F),(C,F),(E,F),(I,J),(G,H),(A,B)(J, K), (F, G), (F, J), (B, F), (C, F), (E, F), (I, J), (G, H), (A, B)(J,K),(F,G),(F,J),(B,F),(C,F),(E,F),(I,J),(G,H),(A,B)

Total Cost: 2+2+3+4+4+5+5+8+9=422 + 2 + 3 + 4 + 4 + 5 + 5 + 8 + 9 = 422+2+3+4+4+5+5+8+9=42.

The reason this method works is because Kruskal's algorithm guarantees an optimal MST, since it always picks the smallest-weight edge available. This reduces the cost to be minimized and avoids cycles. Since we also sorted the edges first we can then process the lightest edges first and stay efficient.

When it comes to time complexity, the sorting of edges takes *O(ElogE)* time, and the Union-Find operations run in nearly *O(1)*, or constant times since the path is compressed. The total complexity comes out to be *O(ElogE)*.

Solution using Prim's Algorithm

Step by step of **Prims's Algorithm**:

1. Start from some arbitrary node of the MST.
2. Using a priority queue (min-heap) we can select the smallest-weight edge that expands the MST.
3. Our repeat step is to add the next lowest-weight edge that connects the MST to a new node.
4. (Recursive) - Repeat until all nodes are included.

The method we used here makes sure MST is connected at all times and efficiently expands the tree. Using a binary heap and adjacency list, the time complexity is O(ElogV), making it very efficient for denser graphs.

Step 1: Choose a Starting Node

Start with A.

Step 2: Expand the MST

- Start at A. The smallest edge is (A,B)(A, B)(A,B) – 9.
- From B, the smallest edge is (B,F)(B, F)(B,F) – 4.
- From F, the smallest edge is (F,G)(F, G)(F,G) – 2.
- From G, the smallest edge is (G,K)(G, K)(G,K) – 5.
- From K, the smallest edge is (J,K)(J, K)(J,K) – 2.
- From J, the smallest edge is (F,J)(F, J)(F,J) – 3.
- From F, the next smallest edge is (C,F)(C, F)(C,F) – 4.
- From F, the next smallest edge is (E,F)(E, F)(E,F) – 5.
- From G, the next smallest edge is (G,H)(G, H)(G,H) – 8.

Final MST Using Prim's Algorithm

Edges in MST (in order of addition):
(A,B),(B,F),(F,G),(G,K),(J,K),(F,J),(C,F),(E,F),(G,H)(A, B), (B, F), (F, G), (G, K), (J, K), (F, J), (C, F), (E, F), (G, H)(A,B),(B,F),(F,G),(G,K),(J,K),(F,J),(C,F),(E,F),(G,H)

Total Cost: 9+4+2+5+2+3+4+5+8=429 + 4 + 2 + 5 + 2 + 3 + 4 + 5 + 8 = 429+4+2+5+2+3+4+5+8=42.

Q1.2: Modified Dijkstra's Algorithm

1. **Initialize**:
    - dist[v] stores the shortest distance from the source node to v.
    - count[v] stores the number of shortest paths to v.
    - Set dist[source] = 0 and count[source] = 1, while initializing all other distances to infinity.
    - Use a priority queue (min-heap) to efficiently fetch the next closest node.
2. **Relaxation Step**:
    - For each neighboring node v of u:
        - If a shorter path is found, update dist[v] and reset count[v] = count[u].
        - If another shortest path is found, add count[u] to count[v], since there's another way to reach v optimally.
3. **Continue Until All Nodes Processed.**

Node	Shortest Distance(dist)	# of Shortest Paths(count)
A	0	1
B	9	1
C	13	1
D	18	1
E	8	1
F	14	1
G	16	1
H	24	1
I	17	1
J	20	1
K	22	1
L	27	1

This works because the algorithm keeps track of the shortest path length while also counting the number of ways to reach each node optimally. Using a priority queue, the complexity is *O((V+E)logV)*.

Q1.3: Modified Bellman-Ford Algorithm

1. **Initialization:**
    - Set dist[S]=0 (distance to the source is 0) and dist[v]=∞ for all other nodes v.
    - Track m[v], the number of edges in the shortest path to v, initialized as ∞ for all v, and 0 for S.
2. **Relaxation Step:**
    - For each edge (u,v) with weight w:
        - If dist[u]+w<dist[v]:
            - Update dist[v]=dist[u]+w.
            - Update m[v]=m[u]+1.
        - Stop the algorithm if m[u]≥m+1, as further passes are unnecessary.
3. **Early Stopping Condition:**
    - Terminate the algorithm after m+1 passes, even if the total number of edges in the graph hasn't been processed.

The standard Bellman-Ford algorithm iterates V−1 times (where V is the number of vertices) to ensure all shortest paths are calculated. However, the shortest path to any node involves at most m edges, so we can stop after m+1 iterations.

Q2: Problem-Solving (Easy)

We need to compute the most **reliable** path in a graph, where each edge has a reliability value between **0 and 1**.

We can modify Djikstra's Algorithm:

1. **Instead of minimizing distance, we maximize reliability**.
2. **Modify Dijkstra's algorithm**:
    - Use rel[v] to store the **maximum reliability** from the source to v.
    - Use a **max-priority queue** (instead of a min-heap).
    - **Relaxation Step**: Update rel[v] = max(rel[v], rel[u] × r(u, v)).

By treating edge reliability as a multiplicative weight, we ensure that paths with higher probabilities are prioritized. The time complexity when using a is O((V+E)logV).

Q3: Adjacency & Weight Matrix (Medium)

Q3.1: Understanding Matrix Powers

Given a graph adjacency matrix M, we define M^2 and M^3.

- **M^2(i,j)=1**: There is a **path of length at most 2** between i and j.
- **M^3(i,j)=1**: There is a **path of length at most 3** between i and j.

This follows from matrix multiplication properties, where each power of M corresponds to paths of increasing length. Because each multiplication step extends the possible paths, it reveals connections through intermediary nodes. When using matrix exponentiation, we can compute M^k in **O(n^3)**

Q3.2: Extra Credit

We have to modify our solution from 3.1 to be usable on weighted graphs:

- Lin Alg 2 Midterm tmrw so no time 🙁 if I submit it late can I still get the extra credit points with 10% off?