CS 677 Homework Assignment 08

Hai Nguyen

December 11, 2018

1. (a) Kruskal's Algorithm: Order that edges are added to the MST:

      - Add (G, H): [G, H], [A], [B], [C], [D], [E], [F], [I]
      - Add (A, C): [A, C], [G, H], [B], [D], [E], [F], [I]
      - Add (I, H): [A, C], [G, I, H], [B], [D], [E], [F]
      - Add (A, B): [A, B, C], [G, I, H], [D], [E], [F]
      - Add (C, D): [A, B, C, D], [G, I, H], [E], [F]
      - Add (C, G): [A, B, C, D, G, I, H], [E], [F]
      - Ignore (B, D): [A, B, C, D, G, I, H], [E], [F]
      - Ignore (D, I): [A, B, C, D, G, I, H], [E], [F]
      - Add (G, F): [A, B, C, D, G, I, H, F], [E]
      - Add (C, E): [A, B, C, D, G, I, H, F, E]

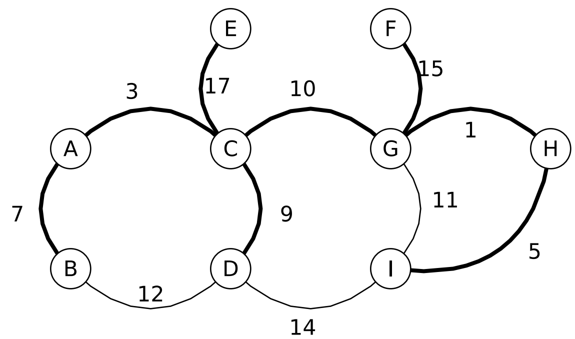

Figure 1: Minimum Spanning Tree.

   (b) Prim's Algorithm: Order that edges are added to the MST, starting from vertex A:

      - (A, C) $\rightarrow$ (A, B) $\rightarrow$ (C, D) $\rightarrow$ (C, G) $\rightarrow$ (G, H) $\rightarrow$ (H, I) $\rightarrow$ (G, F) $\rightarrow$ (C, E)

   Both algorithms yield the same MST as shown in Fig. 1.

2. Page 602, **22.2-9**

   - Apply DFS on the given undirected graph. We know that DFS takes $O(V + E)$ to finish. Also, we know that a DFS forms a depth-first forest which will cover all edges and forms a path along the way. This path will also traverse each edge in E exactly once in each direction because DFS only back-track when it needs to.
   - To find a way out of a maze if we are given a large supply of pennies, we can perform a DFS and use pennies for marking the path that we already traversed through. Each time we traverse through a path we put a penny down and when we back track we put another down. We never traverse in a path with 2 pennies. By this way, we can discover the whole maze and find the way out.

3. Page 621, **22.5-6** The algorithm works as follows:

   - Apply *STRONGLY-CONNECTED-COMPONENTS* algorithm to find all strongly connected components in G and component graph $G^{SCC} = (V^{SCC}, E^{SCC})$: Time $O(V + E)$.

- In each strongly connected component, there will be a directed cycle connected all vertices. Start with $E' = \emptyset$. Time: $O(1)$. For each SCC of G, let the vertices in the SCC be $v_1, v_2, \ldots, v_k$, and add to E' the directed edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)$. These edges form a simple, directed cycle that includes all vertices of the SCC. Time for all SCC's: $O(V)$.
- For each edge (u, v) in the graph G^{SCC}, select any vertex x in u's SCC and any vertex y in v's SCC, and add the directed edge (x, y) to E'. Time: $O(E)$.
- Total time $O(V + E)$.

4. Page 663, **24.3-2**

   Given an example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers.

   Assume that we apply Dijktra's algorithm to the graph with nagative-weight edges as shown in Figure 2. After the aglorithm terminates, we see that the shortest path from the vertice A (the source) to vertice B is 2. However, if we follow the path A → B → C → A → B, the cost is smaller: 2 + 3 - 6 + 2 = 1 < 2.

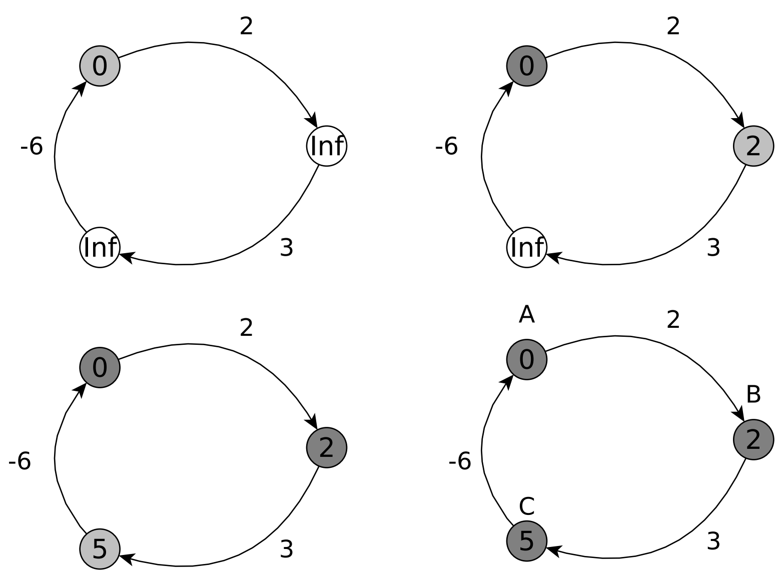

Figure 2: Dijkstra's algorithm on negative-weight edges.

   The proof of Theorem 24.6 fails at (24.2), because it needs to assusme non-negative weight on path $p2$ (Figure 3) in order to have:

$$y.d = \delta(s, y) \leq \delta(s, u) \leq u.d$$

5. Page 692, **25.1-6**

   Suppose that we are given the completed matrix $L(n, n)$ and the weight matrix W, we want to compute the predecessor matrix $\prod$:

   Algorithm:

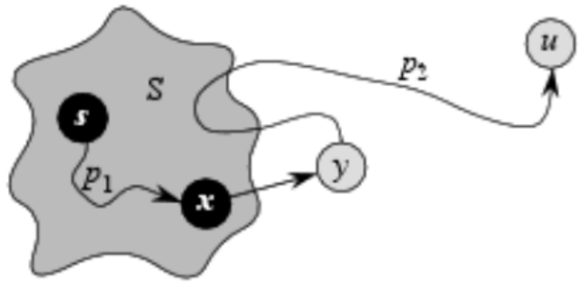

Figure 3: The proof of Theorem 24.6.

COMPUTE-PI(L, W)
for $i = 1 : n$
  for $j = 1 : n$
    for $k = 1 : n$
      if $L(i, j) = L(i, k) + W(k, j)$
        $\prod$(i, j) = k

The given algorithm run in $O(n^3)$ and for a pair of vertices (i, j), it searches all vertices k in between to see if vertice k is in the shortest-path weights from vertice i to vertice j.

6. Page 663, **24.3-6** We build a direct graph G' = (V, E) such that each edge $(u, v) \in E$ the associate weight is $W(u, v) = -\log(r(u, v))$, as shown in Figure 4. As $0 \leq r(u, v) \leq 1$, we have $W(u, v) \geq 0$. By using Dijkstra's algorithm, we can find the path between the two given vertices such that the total weight is smallest.

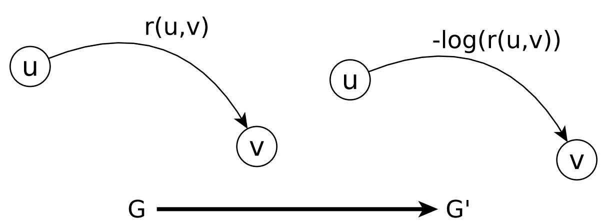

Figure 4: Turning G to G'.

We have:

$$S = \sum_{i=1}^{k} W_i = -\log(\prod_{i=1}^{k} r_i)$$

S is minimum so we have $\prod_{i=1}^{k} r_i$ is maximum, meaning that path is also the most reliable path.