Problem Solving: Homework 3.2

Name: Chen Shaoyuan   Student ID: 161240004

September 11, 2017

1 [TC] Problem 25.1-4

The matrix 'multiplication' defined by EXTEND-SHORTEST-PATHS reads

$$C = A \cdot B$$
$$C_{ij} = \min_{1 \le k \le n} \{A_{ik} + B_{kj}\}$$

To prove the associativity of such 'multiplication', we only have to verify that $(A \cdot B) \cdot C = A \cdot (B \cdot C)$. Let D denote lhs, D' denote rhs:

$$
\begin{aligned}
D_{ij} &= \min_{1 \le l \le n} \{ \min_{1 \le k \le n} \{A_{ik} + B_{kl}\} + C_{lj} \} \\
&= \min_{1 \le l \le n} \{ \min_{1 \le k \le n} \{A_{ik} + B_{kl} + C_{lj}\} \} \\
&= \min_{1 \le k \le n} \{ \min_{1 \le l \le n} \{A_{ik} + B_{kl} + C_{lj}\} \} \\
&= \min_{1 \le k \le n} \{A_{ik} + \min_{1 \le l \le n} \{B_{kl} + C_{lj}\} \} = D'_{ij}
\end{aligned}
$$

this completes the proof of associativity.

2 [TC] Problem 25.1-5

Let W be the adjacency matrix, the single-source shortest-path problem is to calculate $V_i = W^{(\infty)} \cdot W_i$, where W_i denotes the i-th column of W. The index of source is i, and the j-th element of V_i is the weight of the shortest path from i to j. The product of two matrices $C = A \cdot B$ here is defined as

$$C_{ij} = \min_{1 \le k \le n} \{A_{ik} + B_{kj}\}$$

Since a shortest path contains at most $|V| - 1$ edges, we only have to find W raised to the power of $|V| - 1$. We may calculate that from right to left. Each multiplication takes $|V|^2$ time, and we performs such multiplication $|V| - 1$ times, therefore the total running time is $|V|^3$.

3 [TC] Problem 25.1-6

BUILD-PREDECESSOR-MATRIX(W, L, n)

```
1   let Π be a new n × n matrix initialized with NIL
2   for i = 1 to n
3       for j = 1 to n
4           for k = 1 to n
5               if L_ij + W_jk == L_ik and i ≠ k
6                   Π_ik = j
7   return Π
```

Remark: we can prove that, providing only the completed matrix L is not sufficient to compute the predecessor matrix Π. Consider such set of vertex-labeled graph $S_n (n \ge 3)$ that contains all directed n-cycle whose edge weights are zero. Formally speaking, $S_n = \{G_\pi\}$ where π is a permutation of $\{1, 2, \cdots n\}$, and $V(G_\pi) = \{1, 2, \cdots, n\}$, $E(G_\pi) = \{(1, \pi(1)), (2, \pi(2)), \cdots, (n, \pi(n))\}$, and $w(i, j) = 0$ if $(i, j) \in E$. For every $G \in S_n$, G is strongly connected and thus the matrix L_n produced by all-pair shortest-paths algorithm is $0_{n \times n}$. Suppose, to the contrary that it can compute a predecessor matrix Π of L_n, and let $\Pi_{ij} = k$ be any non-nil element of Π, then (k, j) is in one of the shortest-paths trees, and of course it must be in the original graph. However, if we take a permutation π_0 that $\pi_0(j) = k$ and construct a graph G_{π_0}, then $\pi_0(k) \ne j$ and thus $(k, j) \notin E(G_{\pi_0})$, which means the algorithm gives wrong answer.

4 [TC] Problem 25.1-9

FASTER-ALL-PAIRS-SHORTEST-PATHS-MODIFIED(W, n)

```
1   L^(1) = W
2   m = 1
3   while m < 2(n − 1)
4       let L^(2m) be a new n × n matrix
5       L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
6       m = 2m
7   if L^(m) ≠ L^(m/2)
8       error contains negative cycle
9   return L^(m)
```

5   [TC] Problem 25.1-10

MINIMUN-LENGTH-NEGATIVE-CYCLE(W,n)

```
1   L^(1) = W
2   for m = 2 to n
3       let L^(m) be a new n × n matrix
4       L^(m) = EXTEND-SHORTEST-PATHS(L^(m-1),W)
5       for i = 1 to n
6           if L_ii^(m) < 0
7               return m
8   return −1 // does not contain negative cycle
```

The total running time is $O(|V|^2 \cdot ans)$ if the graph contains a negative cycle, or $O(|V|^3)$ if not.

6   [TC] Problem 25.2-2

Let W be the adjacency matrix, where $W_{ij} = 1$ if there exists a directed edge from i to j, or 0 if not. The method of computing transitive closure is the same as 'multiplying matrices' technique described in Section 25.1, except the definition of 'multiplication' of matrices A and B should be changed to

$$C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj}).$$

7   [TC] Problem 25.2-4

The only difference between these two implementations of Floyd-Warshall algorithm is that, when updating d_{ij}, which version of d_{ik} and d_{kj} is used. Note that, during an iteration of outermost loop, for all index i, d_{ik} and d_{ki} are not changed, because $d_{kk} = 0$ and thus $\min(d_{ik}, d_{ik} + dkk) = d_{ik}$ and $\min(d_{ki}, d_{kk} + dki) = d_{ki}$. Therefore, the version of d_{ik} and d_{kj} used for updating d_{ij} does not matter, and the implementation remains correct.

8   [TC] Problem 25.2-6

Let d be the matrix produced by Floyd-Warshall algorithm. If vertex i lies in some negative-weight cycle, then d_{ii} must be negative. So we only have to inspect the diagonal of d. If negative number exists in the diagonal, then the graph must contains a negative cycle.

9   [TC] Problem 25.2-6

TRANSITIVE-CLOSURE(G)

```
1   let T be a new |V(G)| × |V(G)| matrix filled with 0
2   let Vis[1 .. |V(G)|] be a new boolean array
3   for i = 1 to |V(G)|
4       for j = 1 to |V(G)|
5           V[j] = FALSE
6       DFS(T,Vis,G,i,i)
7   return T
```

DFS(T,Vis,G,x,p)

```
1   if Vis[x]
2       return
3   Vis[x] = TRUE
4   T_px = 1
5   for each vertex v in G.Adj[x]
6       DFS(v,p)
```

The procedure described above performs depth-first search from each vertex. Every round of search takes $O(|V| + |E|) = O(|E|)$ time, and it is performed $|V|$ times, so the total running time is $O(|V||E|)$.

10   [TC] Problem 25.3-2

The purpose of adding s to V, is to determining the function $h : V \to \mathbb{R}$, such that after the reweighting described in Lemma 25.1, all edges are non-negative, therefore Dijkstra's algorithm applies. If define $h(v) = \delta(s,v)$, the triangle inequality guarantees the non-negativity of the edges in the new graph.

11   [TC] Problem 25.3-3

If all edge all non-negative in the original graph G, consider the vertex v added to the original graph, since the v is connected to every vertex in G by an edge weighted 0, it is obvious that $\delta(v,u) = 0$ for all u in $V(G)$, i.e. $h(u) = 0$. Therefore, $w = \hat{w}$.

12   [TC] Problem 25-2

a. By referring to Problem 6-2, the asymptotic running times for INSERT, EXTRACT-MIN, DECREASE-KEY are $O(\log_d n)$, $O(d \log_d n)$, $O(\log_d n)$, respectively. If we choose $d = \Theta(n^\alpha)$, their running times are $O(1/\alpha)$, $O(n^\alpha/\alpha)$, $O(1/\alpha)$, respectively. Since the running times for those of a Fibonacci heap are $O(1)$, $O(\log n)$, $O(1)$, respectively, the running

times for INSERT and DECREASE-KEY of a d-ary heap are equal to those of a Fibonacci heap, while d-ary heap is slower in EXTRACT-MIN.

b. Since the graph does not contain negative-weight edge, Dijkstra's algorithm applies here. If we use V^α-ary heap and run Dijkstra's algorithm, the total running time will be $O(V/\alpha + V \cdot V^\alpha/\alpha + V^{1+\varepsilon}/\alpha)$. If we take $\alpha = \varepsilon$, i.e. $d = V^\varepsilon$, since ε is a constant, the total running time is $O(V^{1+\varepsilon}) = O(E)$.

c. For each vertex in the graph, we take it as the source and run Dijkstra's algorithm described above. The total running time is $O(VE)$.

d. We can perform Johnson's algorithm on the graph, and use the above version of Dijkstra's algorithm in Johnson's algorithm. The total running time is $O(VE + VE) = O(VE)$.