Exercise Session 12

Exercise 1.
(CLRS 25.1–4) Show that matrix multiplication defined by EXTEND-SHORTEST-PATH is associative. *Hint:* Let us write $A \odot B$ for EXTEND-SHORTEST-PATH(A, B). You have to prove that for arbitrary $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, and $C \in \mathbb{R}^{p \times q}$ we have that $(A \odot B) \odot C = A \odot (B \odot C)$.

Solution 1.
Consider $(A \odot B) \odot C$. Let $R = (A \odot B)$ and $S = (A \odot B) \odot C$. Then by definition of $\odot$ we have

$$s_{ij} = \min_{1 \le k \le p} \{r_{ik} + c_{kj}\} \qquad \text{and} \qquad r_{ik} = \min_{1 \le l \le n} \{a_{il} + b_{lk}\}\,.$$

Therefore

$$\begin{aligned} s_{ij} &= \min_{1 \le k \le p} \{ \big(\min_{1 \le l \le n} \{a_{il} + b_{lk}\} \big) + c_{kj} \} \\ &= \min_{1 \le k \le p} \{ \min_{1 \le l \le n} \{(a_{il} + b_{lk}) + c_{kj}\} \}\,. && \text{(by distributivity of + over min)} \end{aligned}$$

Now consider $A \odot (B \odot C)$. Let $R' = (B \odot C)$, and $S' = A \odot (B \odot C)$, Then

$$s'_{ij} = \min_{1 \le l \le n} \{a_{il} + r'_{lj}\} \qquad \text{and} \qquad r'_{lj} = \min_{1 \le k \le p} \{b_{lk} + c_{kj}\}\,.$$

Therefore

$$\begin{aligned} s'_{ij} &= \min_{1 \le l \le n} \{a_{il} + \big(\min_{1 \le k \le p} \{b_{lk} + c_{kj}\} \big) \} \\ &= \min_{1 \le l \le n} \{ \min_{1 \le k \le p} \{a_{il} + (b_{lk} + c_{kj})\} \}\,. && \text{(by distributivity of + over min)} \end{aligned}$$

Using the associativity of + we obtain

$$\begin{aligned} s_{ij} &= \min_{1 \le k \le p} \{ \min_{1 \le l \le n} \{(a_{il} + b_{lk}) + c_{kj}\} \} \\ &= \min_{1 \le l \le n} \{ \min_{1 \le k \le p} \{(a_{il} + b_{lk}) + c_{kj}\} \} \\ &= \min_{1 \le l \le n} \{ \min_{1 \le k \le p} \{a_{il} + (b_{lk} + c_{kj})\} \} && \text{(by associativity of +)} \\ &= s'_{ij} \end{aligned}$$

It is concluded that $(A \odot B) \odot C = A \odot (B \odot C)$.

Exercise 2.
(CLRS 25.1–9) Modify FASTER-ALL-PAIRS-SHORTEST-PATHS so that it can determine whether the graph contains a negative-weight cycle. Justify the correctness of your solution.

Solution 2.
Run FASTER-ALL-PAIRS-SHORTEST-PATHS on the graph until the first time that the matrix $L^{(m)}$ has one or more negative values on the diagonal (i.e., $l_{ii}^{(m)} < 0$ for some $1 \le i \le n$), or until we have computed $L^{(m)}$ for some $m > n$. The correctness of the proposed solution is justified by the fact that for any iteration of FASTER-ALL-PAIRS-SHORTEST-PATHS the value $l_{ij}^{(m)}$ is the minimal weight among paths from i to j having at most m edges. Thus $l_{ii}^{(m)} < 0$ represents the minimal weight of a cycle having at most m edges. If a negative cycle exists, it will be found before $m > n$, because any simple path has at most $n - 1$ edges.

Exercise 3.
Let $G = (V, E)$ be a weighted directed graph represented using the weight matrix $W = (w_{ij})$ where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(v_i, v_j) & \text{if } i \neq j \text{ and } (v_i, v_j) \in E \\ \infty & \text{if } i \neq j \text{ and } (v_i, v_j) \notin E \end{cases}$$

How would we delete an arbitrary vertex v from this graph, without changing the shortest-path distance between any other pair of vertices? Describe an algorithm that constructs a weighted directed graph $G' = (V \setminus \{v\}, E')$ such that shortest-path distance between any two vertices in G' is equal to the shortest-path distance between the same two vertices in G in $O(|V|^2)$ time.

Solution 3.
Let $U \subseteq V$ and let $d(W, U)_{ij}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set U. When $U = \emptyset$ we have that $d(W, U)_{ij} = w_{ij}$. For any vertex $v_k \in V$ we have that $d(W, U)_{ij} = \min(d(W, U')_{ij}, d(W, U')_{ik} + d(W, U')_{kj})$, where $U' = U \setminus \{v_k\}$.
Constructing $G' = (V \setminus \{v_k\}, E')$ such that shortest-path distance between any two vertices in G' is equal to the shortest-path distance between the same two vertices in G is equivalent to construct a weighted adjacency matrix W' such that, for all i, j distinct from k we have that

$$d(W', V \setminus \{v_k\})_{ij} = d(W, V)_{ij} \,.$$

One can prove by induction on $|V \setminus \{v_k\}|$ that the above equality holds when $d(W', \emptyset)_{ij} = d(W, \{v_k\})_{ij}$. By applying the definition of d on both sides of the equality we obtain $w'_{i,j} = \min(w_{ij}, w_{ik} + w_{kj})$.
To simplify the exposition we assume that the vertex we want to remove is v_n, i.e., $k = n$. If it is not the case we can simply rearrange the columns and rows of the matrix W so that the vertex k becomes the last one (this can be done in time $\Theta(n^2)$). The following algorithm, given W computes W' as described above after removing the vertex v_n.

```
SkipVertex(W)
  n = W.rows
  let W' = (w'_ij) be a new (n − 1) × (n − 1) matrix
  for i = 1 to n − 1
      for j = 1 to n − 1
          w'_i,j = min(w_ij, w_in + w_nj)
```

One can easily see that the running time of SkipVertex is $O(|V|^2)$.

★ **Exercise 4.**
Assume that $G = (V, E)$ is a directed acyclic graph represented using adjacency-lists.

(a) Describe an algorithm that computes the transitive closure of G, i.e. $G^* = (V, E^*)$, and analyse its running time.

(b) Can you generalise your solution to directed graphs that may contain cycles?

Solution 4.
(a) Let $G^* = (V, E^*)$ be the transitive closure of G. By definition of G^* we have that $(u, v) \in E^*$ if and only if there is a path p in G from u to v, that is $u \rightsquigarrow_p v$. This can be restated as

$$(u, v) \in E^* \text{ if and only if } u = v \text{ or there exists } w \in V \text{ such that } u \to w \rightsquigarrow v \,. \tag{1}$$

If G is a directed acyclic graph we can exploit the fact that we can order its vertices in topological order. Let $v_1, v_2, \ldots, v_n$ be a topological sort for V and let us denote by $G_i = (V_i, E_i)$ the subgraph of G obtained from the subset of vertices $V_i = \{v_i, \ldots, v_n\}$ for $i = 1, \ldots, n$.

For arbitrary $i, j \in \{1, \ldots, n\}$, we rewrite (1) as $(v_i, v_j) \in E^*$ if and only if $i = j$ or $(v_i, v_k) \in E$ and $v_k \rightsquigarrow v_j$ for some k. Note that by the fact that the indices are taken in topological order we can assume that $i < k \leq j$ and that the paths $v_k \rightsquigarrow v_j$ corresponds to an edge in the transitive closure of the subgraph G_{i+1}. Thus, we can obtain the transitive closure of G_i by adding to the transitive closure of G_{i+1} the vertex v_i, the edge (v_i, v_i), and all the edges (v_i, v) such that $(v_i, u) \in E$ and (u, v) is an edge of the transitive closure of G_{i+1}.

The following algorithm implements the above idea maintaining the following invariant for the for-loop in lines 3–5: the subgraph induced by the subset of vertices $\{v_i, v_{i+1}, \ldots, v_n\}$ is the transitive closure of G_i.

DAG-Transitive-Closure(G)

1 let G^* be a new graph with $G^*.V = G.V$ and $G^*.Adj[v] = \{v\}$ for all $v \in G^*.V$
2 let $v_1, \ldots, v_n$ be a topological sort of the vertices of G.
3 **for** $i = n$ **downto** 1
4   **for each** $u \in G.Adj[v_i]$ such that $u \neq v_i$
5         $G^*.Adj[v_i] = G^*.Adj[v_i] \cup G^*.Adj[u]$
6 **return** G^*

Run-time Analysis. Let $|V| = n$ and $|E| = m$. Executing line 1 takes time $\Theta(n)$ and line 2 takes $\Theta(n+m)$. For the running time of the for-loop in lines 3–5 we note that each vertex $u \in G.Adj[v_i]$ represents a vertex in the transitive closure of G_{i+1}, which has $n-i$ vertices, hence $\sum_{u \in G.Adj[v_i]} |G^*.Adj[u]| \leq (n-i)^2$. Therefore, the total number of list insertions performed in the for-loop in lines 3–5 is bounded by

$$\begin{aligned}\sum_{i=1}^{n}(n-i)^2 &= \sum_{k=0}^{n-1} k^2 \\ &= \frac{(n-1)n(2n-1)}{6} \qquad \text{(CLRS Equation (A.3))} \\ &= \Theta(n^3)\end{aligned}$$

Since each list insertion takes constant time, the overall running time of the for-loop is $O(n^3)$. Therefore the running time of DAG-Transitive-Closure is $\Theta(n) + \Theta(n+m) + O(n^3) = O(n^3)$.

(b) The same idea can be applied to generic graph $G = (V, E)$ by noting two things:

(a) the graph G_{scc} of strongly connected components of G is a directed acyclic graph

(b) if u_1 and u_2 belong to the same strongly connected component and $u_1 \rightsquigarrow v$, then also $u_2 \rightsquigarrow v$ (because $u_2 \rightsquigarrow u_1 \rightsquigarrow v$)

Therefore one can compute the transitive closure of G_{scc} as described before extend it to G. The following pseudocode implements the above intuition.

Transitive-Closure'(G)

1 let G^* be a new graph with $G^*.V = G.V$ and $G^*.Adj[v] = \emptyset$ for all $v \in G^*.V$
2 compute the graph G_{scc} of strongly-connected compomponents of G
3 $G^*_{scc} =$ DAG-Transitive-Closure(G_{scc})
4 **for each** $C \in G^*_{scc}.V$
5   **for each** $v \in C$
6         $G^*.Adj[v] = \bigcup G^*_{scc}.Adj[C]$

Exercise 5.

Rick has given Morty a detailed map of the Clackspire Labyrinth, which consists of a directed graph $G = (V, E)$ with non-negative edge weights W (indicating distance from one location in the map to the other), along with a list of dangerous locations $D \subset V$ that Morty has to avoid.

(a) Morty has to determine for each pair of locations $i, j \in V \setminus D$ the length of the shortest walk $i \rightsquigarrow j$ that does not have intermediate vertices in D. Describe the algorithm that Morty can use to solve the problem.

(b) Assume now that Morty also needs to pass by some location in the set $S \subseteq (V \setminus D)$. How does Morty compute the length of the shortest walk from $i \rightsquigarrow j$ that avoids dangerous locations D while ensuring to pass by some location in S?

Solution 5.

(a) Morty can solve the the problem by computing solving the all-pairs shortest-paths e.g., using the Floyd-Warshall algorithm having as input the weighted adjacency-matrix $W = (w_i j)$ defined as follows

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(v_i, v_j) & \text{if } i \neq j \text{ and } (v_i, v_j) \in E \cap (V \setminus D)^2 \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

This corresponds to disconnect all vertices in D form the rest of the graph. Therefore Morty can determine for each pair of locations $i, j \in V \setminus D$ the length of the shortest walk $i \rightsquigarrow j$ that does not have intermediate vertices in D by calling FLOYD-WARSHALL(W). The running time this procedure is $\Theta(|V|^3)$.

(b) Let $D =$ FLOYD-WARSHALL(W) be the matrix obtained before. If we want to ensure that the path passes through some vertex $v_k \in S$, the resulting matrix D' is obtained as $d'_{ij} = \min\{d_{ik} + d_{k,j} \colon v_k \in S\}$. The following pseudocode, implements this idea

```
MORTY(G, D, S)
construct the matrix W as described in Eq.(2)
D = FLOYD-WARSHALL(W)
let D' be a new n × n matrix such that d'_ij = ∞
for each v_i ∈ G.V \ D
    for each v_j ∈ G.V \ D
        for each v_k ∈ S
            d'_ij = min(d'_ij, d_ik + d_k,j)
return D'
```

The running time of MORTY(G, D, S) is $\Theta(|V|^3)$.