CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 11
Submit via Gradescope

Name: Zerun Tian

Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(25 points)***

Following the notes/slides/book, explain All-Source-Shortest-Paths by edges DP using matrix multiplication trick. Write pseudocode for ASSP-Fast and the corresponding Extending-SP procedures.

Solution:

```
1: function ASSP-FAST($W$)                                ▷ $W$ is a set of edge weights
2:     $m = 1$
3:     $C^{(m)} = W$
4:     while $m < |V| - 1$ do
5:         $m = 2 \cdot m$
6:         $C^{(m)} =$ EXTENDING-SP($C^{(m/2)}, W$)
7:     return $C^{(m)}$
```

Here is the modified EXTENDING-SP procedure,

```
1: function EXTENDING-SP($C$, $W$)
2:     for $i = 1$ to $|V|$ do
3:         for $j = 1$ to $|V|$ do
4:             $a = \infty$
5:             for $k = 1$ to $|V|$ do
6:                 $a = \min(a, C_{ik}^{(m/2)} + C_{kj}^{(m/2)})$
```

The matrix multiplication trick (repeated squaring) works for the ASSP problem because $C^{(m)}$ can be derived from $C^{(m/2)}$ which is a solution to $m/2$ number of edges. For example, we can solve $C^{(16)}$ by using $C^{(8)}$. First of all, $C^{(8)}$ gives us the shortest path costs between all pairs of nodes i, j within 8 edges. Then, for paths between i and j, we can find the shortest path within 16 edges by considering two paths each of which are within 8 edges. The combination that has the lowest cost wins. Formally, it can be expressed as $C_{ij}^{(m)} = \min_{1 \le k \le n} \left\{ C_{ik}^{(m/2)} + C_{kj}^{(m/2)} \right\}$.

Furthermore, notice that in ASSP-FAST, m might eventually surpasses $|V| - 1$, which is not a problem. This is because, in a graph with $|V|$ nodes, any shortest path will be found within $|V| - 1$ edges in $C^{(|V|-1)}$. Any extra edges won't affect the shortest path outcome beyond $|V| - 1$. Thus, $C^{(m)}$ would be identical to $C^{(|V|-1)}$, where m is the smallest power of 2 that is $\ge |V| - 1$.

Thanks to the trick, ASSP-FAST is able to output a solution in $O(V^3 \lg V)$ as m will be greater than or equal to $|V| - 1$ after $\lg V$ iterations.

2. ***(25 points)*** *Exercise 24.1-3.*

Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

Solution:

```
 1: function BELLMAN-FORD($G$, $w$, $s$)
 2:     INITIALIZE-SINGLE-SOURCE($G, s$)
 3:     for $i = 1$ to $|G.V| - 1$ do
 4:         $changed$ = FALSE
 5:         for each edge $(u, v) \in G.E$ do
 6:             $vd = v.d$
 7:             RELAX($u, v, w$)
 8:             if $vd \neq v.d$ then                    ▷ estimate for node $v$ is updated
 9:                 $changed$ = TRUE
10:         if not $changed$ then
11:             break
12:     for each edge $(u, v) \in G.E$ do
13:         if $v.d > u.d + w(u, v)$ then
14:             return FALSE
15:     return TRUE
```

To achieve early stopping, we add some lines of code to the original BELLMAN-FORD. At line 4, we initialize a flag *changed* that will be set to TRUE if any node's estimate d is changed. Otherwise, the flag will remain FALSE during the execution of the for loop in lines 5-9 which causes the for loop in lines 3-11 to break. Essentially, we detect a sign of convergence in computing the shortest path where no relaxation is made after trying to relax all edges. The algorithm will reach the described scenario in the m-th "global" relaxation (in the m-th iteration of the for loop at line 3). It will then realize it in the $m+1$ pass by checking that no change is made to any node's estimate, and terminate.

3. ***(25 points)*** *Exercise 24.2-2.*

Suppose we change line 3 of DAG-SHORTEST-PATHS to read

```
3  for the first $|V| - 1$ vertices, taken in topologically sorted order
```

Show that the procedure would remain correct.

Solution:

Because we are dealing with a DAG that is topologically sorted on a line, we know there is no outgoing edge from the last vertex (the $|V|$-th vertex). Otherwise, it would form a cycle which contradicts that it's a DAG. In the unmodified DAG-SHORTEST-PATHS, the for loop in lines 4-5 won't execute when u is the last vertex because its adjacency list is empty. Thus, the change of line 3 will not make a difference in the outcome. The procedure stays correct.

4. ***(25 points)*** *Exercise 24.3-4.*

Professor Gaedel has written a program that he claims implements Dijkstra's algorithm. The program produces $v.d$ and $v.\pi$ for each vertex $v \in V$. Give an $O(V + E)$-time algorithm to check the output of the professor's program. It should determine whether the d and π attributes match those of some shortest-paths tree. You may assume that all edge weights are nonnegative.

Solution:

First of all, we make sure the algorithm indeed produces a tree. Since each vertex has only one parent pointed through $v.\pi$, we know the output is a tree by checking two things: only one vertex s does not have a parent; vertices in $V - \{s\}$ are not self-loops (parents are not themselves). We can quickly verify this by iterating through all the vertices and check if ($v.\pi \neq \text{NIL}$ and $v.\pi \neq v$) for $|V| - 1$ vertices, and $v.\pi == \text{NIL}$ for only 1 vertex who is the source. This step takes $O(V)$-time.

Now that we can verify if the algorithm produces a tree, we want to check if it produces a shortest path for every vertex. To do that, we simply relax all edges in the resulting graph once. If any relaxation happens to lower the d value for some vertex v, it implies that the professor's algorithm does not manage to find a shortest path for that vertex. This step takes $O(E)$ time to relax all edges.

Moreover, there are some small things we need to take care of. According to the Dijkstra's algorithm, we want to verify $s.d$ is 0. In addition, we want to know if $v.\pi.d + w(v.\pi, v) = v.d$ for each vertex $v \in V - \{s\}$. This can be done in $O(V)$-time by checking if the equality holds for all vertices except the source.

While applying these procedures, if the output of the professor's program fails at any stage, the program is incorrect. Otherwise, we claim the program is correct in $2 \cdot O(V) + O(E) + O(1) = O(V + E)$ time.

5. ***(Extra Credit)*** *Problem 24-2.*

Solution:

6. ***(30 points)*** *Explain in few lines the concept of transitive closure.*

Solution:

A transitive closure can answer whether or not there is a path from vertex i to j in a graph G. Specifically, the transitive closure of a graph $G = (V, E)$ is defined as a graph $G^* = (V, E^*)$ whose vertices are identical to the original graph G; and E^* is a superset of E. This is because for every edge $(i, j) \in E$, it is also a path $i \rightsquigarrow j$ in E^*. Thus, we can think of the transitive closure as an adjacency matrix where the entry $[i, j] = 1$ if there exists a path between i and j, and $[i, j] = 0$ otherwise.

We can compute the transitive closure of G by slightly modifying the FLOYD-WARSHALL algorithm. Initially, we initialize a matrix T whose is the adjacency matrix of G. Note that $T[i, j]$ is also set to 1 if $i = j$. Moreover, in the most inner loop, we substitute the statement of **min** and **+** operators with boolean operators $\vee$ and $\wedge$. Specifically, we want to set $T_{ij}^{(k)}$,

$$T_{ij}^{(k)} = T_{ij}^{(k-1)} \vee (T_{ik}^{(k-1)} \wedge T_{kj}^{(k-1)})$$

What this means is that there will be a path from vertex i to j within k intermediate vertices if there is either a path within $k - 1$ intermediaries already OR a path passing through the vertex k when k is reachable from i, AND j is reachable from k indicated by $T_{ik}^{(k-1)}$ and $T_{kj}^{(k-1)}$, respectively. The algorithm outputs the transitive closure of G after the outer loop runs n iterations, which covers paths of n vertices. The runtime is also $\Theta(n^3)$ but slightly better than solving the S.P problem in practice within some constant factor.

7. ***(20 points)*** *Exercise 25.1-6 (the book uses different notation for the matrices). Also explain how to use this result in order to display all-pair shortest paths, enumerating intermediary vertices (or edges) for each path.*

Suppose we also wish to compute the vertices on shortest paths in the algorithms of this section. Show how to compute the predecessor matrix $\prod$ from the completed matrix L of shortest-path weights in $O(n^3)$ time.

Solution:

For each cell (i, j) of the matrix $\prod$, we find the predecessor $k = \min_k \left\{ C_{ik} + w_{kj} \right\}$. The predecessor is an intermediary vertex k who has an outgoing edge $k \to j$ given that the path from i to j through the $k \to j$ edge presents the minimum cost.

```
function COMPUTE-PREDECESSOR-MATRIX($C$, $w$)
    init $\prod$ as a matrix of size $|V|$ by $|V|$
    for $i = 1$ to $|V|$ do
        for $j = 1$ to $|V|$ do
            if $i == j$ then
                $\prod[i, j] = $ NIL
            else
                for $k = 1$ to $|V|$ do
                    if $C[i, k] + w(k, j) == C[i, j]$ then
                        $\prod[i, j] = k$
    return $\prod$
```

We can enumerate intermediary vertices for each path using the predecessor matrix $\prod$ as follows,

```
function PRINT-INTERMEDIARY-VERTICES($\prod$)
    for $i = 1$ to $|V|$ do
        for $j = 1$ to $|V|$ do
            $path = [j]$                         ▷ an empty array to store the S.P. from $i$ to $j$
            $p = \prod[i, j]$                    ▷ keeps track of the predecessor
            while $p \neq$ NIL do
                $path$.append($p$)
                $p = \prod[i, p]$
            for $k = path$.length to 1 do        ▷ prints the vertices in the S.P. from $i$ to $j$
                PRINT($path[k]$)
```

*Note that we use the notation learned from lectures, specifically using C instead of L to represent the shortest path matrix.

8. ***(20 points)*** *Exercise 25.2-1.*

Run the Floyd-Warshall algorithm on the weighted, directed graph of Figure 25.2. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.

Solution:

The initial $D^{(0)}$ is identical to W of the graph. Since there are 6 nodes in the graph, the outer for loop has 6 iterations. Here is the matrix $D^{(k)}$ after each iteration,

$$D^{(0)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}, D^{(1)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}, D^{(3)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}, D^{(5)} = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^{(6)} = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

9. ***(20 points)*** *Exercise 25.2-4.*

As it appears above, the FLOYD-WARSHALL algorithm requires $O(n^3)$ space, since we compute $d_{ij}^{(k)}$ for $i, j, k = 1, 2, \cdots, n$. Show that the following procedure, which simply drops all the superscripts, is correct, and thus only $O(n^2)$ space is required.

```
1: function FLOYD-WARSHALL'(W)
2:     $n = W.rows$
3:     $D = W$
4:     for $k = 1$ to $n$ do
5:         for $i = 1$ to $n$ do
6:             for $j = 1$ to $n$ do
7:                 $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$
8:     return D
```

Solution:

Comparing this with the original algorithm, we are modifying the D matrix in place at every iteration. We are updating the D matrix cell by cell where the new value d_{ij} is dependent on two things:

First, it depends on the old value $d_{ij}^{(k-1)}$ in that cell which will not be modified by iterations prior to the i-j iteration of the for loop in lines 5-7.

Second, it depends on the $d_{ik} + d_{kj}$ value after the $k-1$ iteration of the outer for loop. We claim that both d_{ik} and d_{kj} are unmodified. We observe that the k-th row of $D^{(k)}$ would be identical to that of the $D^{(k-1)}$. This is because at the k-th iteration of the outer for loop, we have that

$$d_{kj}^{(k)} = \min(d_{kj}^{(k-1)}, d_{kk}^{(k-1)} + d_{kj}^{(k-1)}) = \min(d_{kj}^{(k-1)}, 0 + d_{kj}^{(k-1)}) = d_{kj}^{(k-1)}$$

Similarly, we observe that the k-th column of $D^{(k)}$ is identical to that of the $D^{(k-1)}$. For a similar reason,

$$d_{ik}^{(k)} = \min(d_{ik}^{k-1}, d_{ik}^{(k-1)} + d_{kk}^{(k-1)}) = \min(d_{ik}^{k-1}, d_{ik}^{(k-1)} + 0) = d_{ik}^{(k-1)}$$

Because the d_{ij} update depends on things that won't be modified at all between the two iterations of the outer for loop, the in-place variant of FLOYD-WARSHALL is correct.

10. ***(Extra Credit)*** *Exercise 25.2-6.*

Solution: