CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 11
Submit via Gradescope

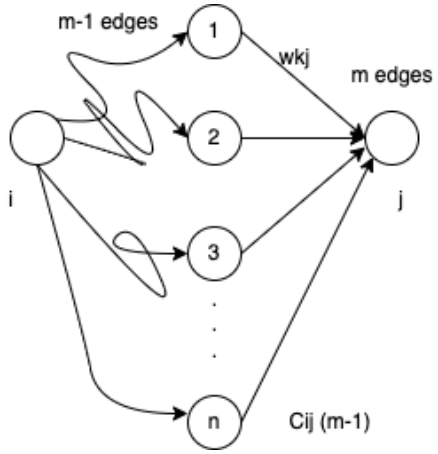
Name: Ketaki Kolhatkar
Collaborators:

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:



ASSP uses dynamic programming to calculate the shortest path from the source to the destination by calculating all the paths and then selecting the minimum one from those. We use the bottom up approach for this computation. Here, we first calculate all the path from i to k (intermediary), that is for m-1 edges and then add it to the paths from k to the destination which is j here. We then select the one which gives us the minimum value from these. We calculate this using the formula $min(C_{ij}^{m-1}, C_{ij}^{m-1} + w_{kj})$.

```
Extend-SP(Cm-1, W)
    for i = 1:n
        for j = 1:n
             a = infinity
             for k = 1:n
                a = min{a, Cik(m-1) + wkj}
            Cij(m)=a
```

This looks similar to matrix multiplication, where we use the formula $a = a + Cik * wkj$ instead of the one which we use in Extend-SP. Both the a calculations take O(1) time and hence we can consider them equivalent.
We can calculate the C_{ij} as follows :
$C^{(1)} = C^{(0)} * W = W$,
$C^{(2)} = C^{(1)} * W = W1$
and so on.
Here, to reduce the runtime, we only calculate the $C^{(n-1)}$, which is $C^{(1)}$, $C^{(2)}$, $C^{(4)}$, $C^{(8)}$... This reduces our time complexity to only O(logn). The previous runtime of $O(n^4)$ is reduced to $O(n^3 logn)$.

```
ASSP-FAST(W)
    C(1) = W
```

```
    while m<n-1
        C(m) = Extend-SP(C(m-1), C(m-1), W)
        m = 2 * m
    return C(m)

D = Multiply(C, W)  #Corresponding SP
    for i = 1:n
        for j = 1:n
            a = 0
            for k = 1:n
                a = a + Cik * wkj
            Dij = a
```

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

Solution:
According to the Bellman Ford algorithm, once m (total number of edges on the shortest path) iterations have taken place, there won't be any updating on the values of the vertices, that is they must have reached the minimum value. Hence for the m+1th iteration, no value of any vertex will change, and it is expected that they remain constant. Therefore, even if we do not know the value of m, we can track the value of v at every iteration to see if it was updated. Once it stops updating, we can conclude that we have reached the m+1th iteration and stop the algorithm.

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

Solution:
Topologically ordering means that the vertex with the earliest finish time is sorted at the end and the one with the latest finishing time is at the first. There would be edges from the highest one and the following vertices only in the right hand side direction. This shows the dependencies. But this also means that the last vertex, will not have any edge pointing from itself to another vertex since that is the right most vertex. Considering this, even if we consider only —V—-1 vertices, that would not make a difference since the algorithm does not proceed after line 3 for the last vertex. Hence, the procedure would still remain correct.

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

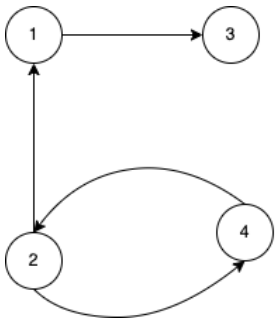
Solution:
The relaxed d values and pi values forming the shortest path can be used to verify the shortest path between the source and the destination. We can use intuition followed in the SSSP algorithm where all the vertices are topologically sorted. The runtime of this would be $O(V + E)$. When we are going through the loop of the sorted order of vertices, we would check if the source s's s.d is equal to zero and s.pi is NIL. This is because the source - first element can't have any vertex before it at the start of the graph execution. Next we check if $v.d = v.\pi.d + w(v.\pi, v)$ where u = $v.\pi$, and v is any other node than s. Considering that all the cases given are being followed, then we can say that v.d is the minimum cumulative weight, and p is the shortest path to v. If the conditions are not followed, then we can conclude that the shortest path found is incorrect. The overall time complexity obtained would be $O(V + E)$.

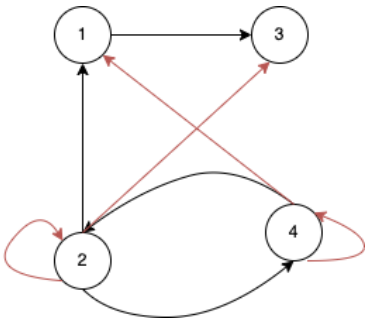
5. ***(Extra Credit)*** *Problem 24-2.*

Solution:

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

Solution:





Consider the DAG given on the left hand side. We get a matrix A considering the vertices that are connected with one another with the given edges. Matrix A would be = $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$. But, if we take a closer look at the graph and also consider the vertices that are connected to one another indirectly through other vertices, we would get a different matrix A. Considering vertex 1, we see that it is just connected to 3 and there is no other outgoing edge for it to be indirectly connected to another edge. Considering edge 2, we see that 2 has an edge going towards 1 and 1 is outgoing to 3. This means that, 2 is indirectly connected to 3. Another outgoing edge of 2 is to 4, which is coming back to 2. This means that is forming an edge with itself indirectly. Hence we change the value of row 2 column 2 and column 3 to 1. Similarly, we check the indirect dependence of each vertex on another and change the matrix A accordingly. This indirect relation is called the transitive closure. The changed matrix A would look like : $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$. We solve this problem using dynamic programming by the Floyd Warshall's algorithm. Here, we consider the previous matrix and try to solve ahead. If we come across a 1 in the matrix, we leave the number as is, else if we get a 0, we check if there any intermediary path from i to k and k to j, then we can say that there is a path between i to j, and change the 0 value to 1.

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.*

Solution:

We need to compute the shortest path for each vertex. For a specific i and j, the value of k would

such that $L_{i,k} + w(k, j) = L_{i,j}$. We would take a time complexity of $O(n^3)$ since we need to compute i, j, k each of which require a time complexity of O(n).

```
PATH-(L, w)
    for i = 1:n
        for j = 1:n
            for k = 1:n
                do if L(i,k) +w(k, j) = L(i,j)
                    do (i,j) = k
```

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

Solution:

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

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

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

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

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

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

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

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

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

Solution:

While following the e Floyd-Warshall algorithm we require a space complexity of $O(n^3)$ since for every iteration of k we fill in the updated values of i and j. We can reduce the space required to $O(n^2)$ by just updating the matrix by the value that is changed for every k iteration. This will give us a 2D matrix instead of a 3D one. The procedure remains the same, the only difference we have is that instead of storing the values for every k iterations, we update it in the same matrix for the k iterations.

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

Solution: