CS 374 HW 7 Problem 2

Aldo Sanjoto, quddus2, Hieu Huynh

TOTAL POINTS

80 / 100

QUESTION 1

1 2A 20 / 20

✓ **- 0 pts** Correct

- 20 pts Adding Every edge possible

- 10 pts Adding sub-optimal number of edges in linear time

- 10 pts Adding optimal number of edges in non-linear time

- 5 pts Doesn't work for disconnected DAGs

- 2 pts Number of edges added is wrong/not present

- 15 pts IDK

- 20 pts Illegible/Lengthy

QUESTION 2

2 2B 0 / 20

- 0 pts Correct

- 10 pts Non-linear time algorithm

✓ **- 5 pts Not mentioning the sub-optimality of their algorithm**

✓ **- 15 pts Used DFS or BFS**

- 20 pts Unclear/illegible/lengthy

- 2 pts Minor error

- 2 pts Minor Error

- 20 pts Incorrect algorithm

- 15 pts IDK

Solutions using BFS or DFS get 5 points. These algorithms would be incorrect because the nodes could have multiple parents and if along a different parent, the number of important vertices increases, you will have to update it all along the path, which means the algorithm is not linear anymore. Extra 5 points are cut for not noticing this bug.

QUESTION 3

3 2C 30 / 30

✓ **- 0 pts** Correct

- 10 pts non-linear time solutions if state the correct runtime

- 15 pts non-linear time solutions if not state the correct runtime

- 30 pts unclear or way too lengthy answers

- 22.5 pts IDK

QUESTION 4

4 2D 30 / 30

✓ **- 0 pts** Correct

- 3 pts Added edges from algorithm from Part A should have weight/length 0.

- 5 pts Should assign all original edge weights to 1 (or equivalently, all original edge weights should be the same positive value) in order to run the algorithm from Part C

- 10 pts Assuming a unique source in the graph

- 30 pts Unclear / Incorrect Algorithm

- 2 pts Must explicitly state the algorithm's runtime even if using the algorithm from Part C.

- 22.5 pts IDK

- 15 pts Non linear algorithm

Q2

A.) • Do topological sorting of the list of verticies } $O(n+m)$

• for $i=1$ to n:

get outgoing edges of v_i

for each $(v_i, v_j) \in E$, mark v_j is not source } $O(n+m)$

• for $i=1$ to n:

if ($v_i \neq s$ & v_i is a source):

Add edge: (s, v_i) } $O(n)$

Total $O(n+m)$ times

# of edges: let k be the # of sources in orginal graph then add $(k-1)$ edges

---

B.) DFS(u, count){

Mark u as visted

for

1 2A 20 / 20

✓ - 0 pts Correct

- 20 pts Adding Every edge possible
- 10 pts Adding sub-optimal number of edges in linear time
- 10 pts Adding optimal number of edges in non-linear time
- 5 pts Doesn't work for disconnected DAGs
- 2 pts Number of edges added is wrong/not present
- 15 pts IDK
- 20 pts Illegible/Lengthy

2b.)

Algorithm: result // array of verticies

```
Somefunc (u, count, result){
    for each edge (u,v) in Out (u) {
        if v is not visited{
            Mark v as visited
            if (V is important){
                count++;
            }
            If (count ≥ τ){
                add v to result
            }
            DFS(v, count)
        }
    }
}
```

To get all vertices satisfy the condition,

do: somefunc (s, 0, result)

2 2B 0 / 20

- **- 0 pts** Correct
- **- 10 pts** Non-linear time algorithm
- ✓ **- 5 pts Not mentioning the sub-optimality of their algorithm**
- ✓ **- 15 pts Used DFS or BFS**
- **- 20 pts** Unclear/illegible/lengthy
- **- 2 pts** Minor error
- **- 2 pts** Minor Error
- **- 20 pts** Incorrect algorithm
- **- 15 pts** IDK

Solutions using BFS or DFS get 5 points. These algorithms would be incorrect because the nodes could have multiple parents and if along a different parent, the number of important vertices increases, you will have to update it all along the path, which means the algorithm is not linear anymore. Extra 5 points are cut for not noticing this bug.

2c)

- Do DFS(s) to find all vertices that can be reached from s
- Do Topological sorting on these vertices and store the result in array A [1...n] with n be the number of vertices in reach(s)

- Let array Distance [1...n] store the longest path from s to vi with vi be the vertex at index i in A[1...n]
- s = A[1] => Distance [1] = 0
- Initialize all element in Distance array to $-\infty$ (except the Distance[1])
- For i ← 1 to n {

      For each $v_j \in Out(v_i)$ {
          Distance[v_j] = max{Distance[v_j], distance[v_i] + $\ell(v_i, v_j)$}
      }
  }

Note: $\ell(v_i, v_j)$ is length of edge (v_i, v_j)

This algorithm takes O(m+n).

because doing DFS(s) take O(m+n)

Topological sort take O(m+n)

and the loop take O(n)

3 2C 30 / 30

✓ - 0 pts Correct

- 10 pts non-linear time solutions if state the correct runtime
- 15 pts non-linear time solutions if not state the correct runtime
- 30 pts unclear or way too lengthy answers
- 22.5 pts IDK

2D)

n = total number of verticies

- let longestpath array have n elements & initialize to 0
  Successor array have n elements & initialize to null $\Big\}$ $O(n)$

- Topological sort the verticies $\}$ $O(m+n)$

- for $i \leftarrow n$ to 1;

```
for vj ∈ IN(vi) {
        if longestpath[Vj] < longestpath[Vi] +1 {
                longestpath[Vj] = longestpath[Vi] +1
                Successor[Vj] = Vi
        }
}
```

$O\left(n + \sum_{\substack{i=1 \\ v_i \in vertex}}^{n} deg(v_i)\right)$

$\Downarrow$

$O(n+m)$

- find max element in longestpath

- V = vertex that is the max element in longestpath

```
While ( V != sinker ):
        Print (v);
        V = Successor[v]
```

$\Big\}$ $O(n)$

Worst case running time is $O(n+m)$

4 **2D** **30 / 30**

✓ **- 0 pts** Correct

- 3 pts Added edges from algorithm from Part A should have weight/length 0.

- 5 pts Should assign all original edge weights to 1 (or equivalently, all original edge weights should be the same positive value) in order to run the algorithm from Part C

- 10 pts Assuming a unique source in the graph

- 30 pts Unclear / Incorrect Algorithm

- 2 pts Must explicitly state the algorithm's runtime even if using the algorithm from Part C.

- 22.5 pts IDK

- 15 pts Non linear algorithm