CS180 HW3

Siddharth Joshi

TOTAL POINTS

100 / 100

QUESTION 1

1 Problem 1 25 / 25

✓ **- 0 pts Correct**

- 5 pts Preprocessing not correct

- 5 pts failed to move two pointers together

- 10 pts error in algorithm; or algorithm runtime not correct

- 20 pts wrong answer but showed efforts

- 25 pts no answer

QUESTION 2

2 Problem 2 25 / 25

✓ **- 0 pts Correct**

- 5 pts did not update queue during the loop

- 10 pts failed to reduce the original problem

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer or no answer

QUESTION 3

3 Problem 3 25 / 25

✓ **- 0 pts Correct**

- 3 pts Complexity exceeds O(E)

- 4 pts Fail to justify answer or need more explanation

- 4 pts Fail to show the algorithm step-by-tep

QUESTION 4

4 Problem 4 25 / 25

✓ **- 0 pts Correct**

- 4 pts fail to justify answer

- 4 pts fail to show algorithm step-by-step

HW 3

1. Let paths be a hashmap from vertices to a vector of vertices where every vertex is the key for the path that leads to it from the root vertex

```
preProcess (root).
    vector S = [root]
    mark root as traversed
    while (S is not empty):
        paths [S[length of s - 1]] = S
        if S[length of s - 1] has any untraversed children:
            let v be one such vertex
            mark v as traversed
            append v to S
        else:
            delete & remove the last element of S from S

LowestCommonAncestor ( u, v).   // Assume preprocessing has
  let A = paths [u]             // been done
  let B = paths [v]
  int i = length(A) - 1     int j  length (B) - 1
  let set X and set Y be 2 empty sets
  while ( i >= 0 and j > 0).
      if A[i] is not in Y and A[i] != B[j]
            add A[i] to X and decrement i
      else: return A[i]
      if B[j] is not in X then add B[j] to Y and decrement j
       else: return B[j]
```

contd. on next page

```
while (i > 0):
    if A[i] is not in Y.
        add A[i] to X and decrement i
    else return A[i]

while (j >= 0):
    if B[j] is not in X:
        add B[j] to X and decrement j
    else return B[j]
```

Time complexity: the preprocessing function is simply an iterative implementation of DFS using the vectors as a stack-of sorts since T is a tree $|E| = |V| - 1$ $\therefore O(|E|) = O(|V|)$ and hence DFS ~~become~~ that is typically $O(|V|+|E|) = O(|V|+|E|) = O(|E|)$ The lowest Common Ancestor function then has loops that run for a total of $\max(i,j)$ iterations, but since $\max(i,j) \leq h$ where h is the height of the tree LowestCommonAncestor has runtime $O(h)$

Correctness: The preprocessing computes the correct path to each vertex as at any point in DFS the stack has the vertices used to get to the current vertex and since vector S is a defacto stack $\Rightarrow$ correct. Also all vertices are traversed as DFS does thus $\therefore$ preprocessing is correct. The lowest Common Ancestor function is correct as it traverses the path vectors for u and v in reverse order and returns the 1st common element i.e the first common ancestor from the back $\Rightarrow$ the lowest common ancestor. Since the path includes the vertex itself i.e. path to u contains u and since it contains root, the edge cases where u is a child of v (or vice-a versa) and LCA(u,v) root are also correct

1 Problem 1 **25 / 25**

✓ **- 0 pts Correct**

- 5 pts Preprocessing not correct

- 5 pts failed to move two pointers together

- 10 pts error in algorithm; or algorithm runtime not correct

- 20 pts wrong answer but showed efforts

- 25 pts no answer

2. Directed

largestSubsetS (Graph G where vertices $v_0, \ldots, v_{n-1}$ represent the n cities and there exists an edge (v_i, v_j) if and only if there is a flight from v_i to v_j):

```
let inDegree be a vector of size n initialized to all 0s
for each edge (src, dest) in G:
    inDegree[dest]++

while inDegree of some vertex = 0:
    for every vertex v_i with inDegree[v_i] 0:
        for every edge (v_i, v_j):
            inDegree[v_j]--

let S be an empty set
for every vertex v_i in G:
    if inDegree[v_i] ~~==~~ !. 0
        add ~~it~~ v_i to S
return S
```

Correctness: The algorithm makes at most n passes over the set of vertices and in every pass deletes makes vertices with 0 incoming edges from the current set of possible members of S, ineligible to i.e. they are no longer possible members of S. In this way it can be thought that the set of all vertices is reduced iteratively till or all the vertices left have an incoming edge from other vertices left.

Time Complexity: Since the algorithm essentially traverses over the set of vertices V and in each iteration 'deletes' in some sense n vertices (~~traversing in the iteration~~)

(terminating in the iteration that $x = 0$), the worst case is when in each pass only 1 vertex is 'deleted' and all vertices must be 'deleted'

∴ complexity of the nested while and for loops $O(n^2)$

The construction of the initial inDegree vector takes $O(|E|)$

but since $|E| = n$ (given), the total time complexity is

$O(n^2 + n) \leftarrow \boxed{O(n^2)}$

2 Problem 2 25 / 25

✓ **- 0 pts** **Correct**

- 5 pts did not update queue during the loop

- 10 pts failed to reduce the original problem

- 20 pts wrong answer but showed efforts

- 25 pts wrong answer or no answer

3.

determineACyclic (Directed graph G (V, E)).

⇒ let V be a set = V the set of vertices of the graph

⇒ let current be any vertex in V

⇒ let T be an empty set

⇒ let S be an empty stack

⇒ push current on to S and remove current from V

⇒ while (S is not empty):

→ let u = S.top()

if there exists an edge (u, w):

- if w is in T:
  return True i.e. G has a cycle
- else:
  delete (u, w)
  push w onto the stack S
  remove w from V

→ else:

pop (S) // pop u off the stack

add u to T

if (S is empty and V is non empty)

push
add a vertex in V onto S and remove it from V

→ return False i.e. G is acyclic

Time Complexity: The while loop essentially executes DFS which is typically O(|V|+|E|) but since it is known that there is atleast one edge (incoming or outgoing) for every vertex v. ⇒ $|E| \geq \frac{|V|}{2}$ ∴ O(|E|) ≥ O(|V|) and hence the algorithm has O(|E|) runtime

Correctness · The algorithm essentially 'tries' to topologically sort G where T indicates the set of vertices that have been sorted and ∴ if the algo tries to essentially readd a vertex to T i.e. ~~add~~ tries to topologically sort a vertex that has already been sorted then $\Rightarrow$ G has a cycle and hence the algorithm terminates indicating this. If the end of the while loop is reached i.e ~~some~~ ~~to~~ all vertices have been topologically sorted $\Rightarrow$ G is acyclic as only acyclic graphs can be sorted as such and hence the algorithm indicates this and terminates

3 Problem 3 25 / 25

✓ **- 0 pts** **Correct**

- 3 pts Complexity exceeds O(E)

- 4 pts Fail to justify answer or need more explanation

- 4 pts Fail to show the algorithm step-by-tep

4.

topologicalSortDFS (A directed acyclic graph G (V, E)):

⇒ let T be an empty vector

⇒ let S be an empty stack

⇒ push any vertex $v \in V$ onto S

⇒ while (S is not empty):

→ let u = S.top()

→ if there exists an edge (u, v) where v is an unmarked ~~untraversed~~ vertex:

• ~~add~~ push v on to S and continue to next iteration

→ else:

• pop (S) // pop u off of stack S

• prepend u to T

• mark u

→ if (S is empty and there exists a vertex ~~in~~ • $w \in V$ s.t. w is unmarked):

• push w on to stack S

⇒ return T // the vector indicating a topological sorting of G

Time Complexity: The while loop essentially carries out DFS which typically has time complexity $O(|V| + |E|)$ but since it can be assumed that every vertex has atleast 1 edge $|E| \geq |V|/2 \Rightarrow O(|E|) \geq O(|V|)$ $\therefore O(|V| + |E|) \leq O(|E|)$ ∴ this algorithm has $O(|E|)$ time complexity

Correctness: At each step a vertex is prepended to T if and only if it has no unmarked children left ⇒ all its children/neighbours are already correctly in T or if it has none, either way the new vertex is thus correctly added to T and DFS implemented in the way it is above ensures every vertex $v \in V$ is traversed. Therefore algorithm is correct

4 Problem 4 25 / 25

✓ **- 0 pts** **Correct**

- 4 pts fail to justify answer

- 4 pts fail to show algorithm step-by-step