Question 1 (15 points). *Suppose you are given a directed graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5, 7, 8\}$ and the depth first intervals ([pre, post]) of each vertex are as follows.* $\{1 : [4, 5], 2 : [7, 8], 3 : [12, 13], 4 : [14, 15], 5 : [3, 6], 6 : [2, 9], 7 : [1, 10], 8 : [11, 16]\}$.

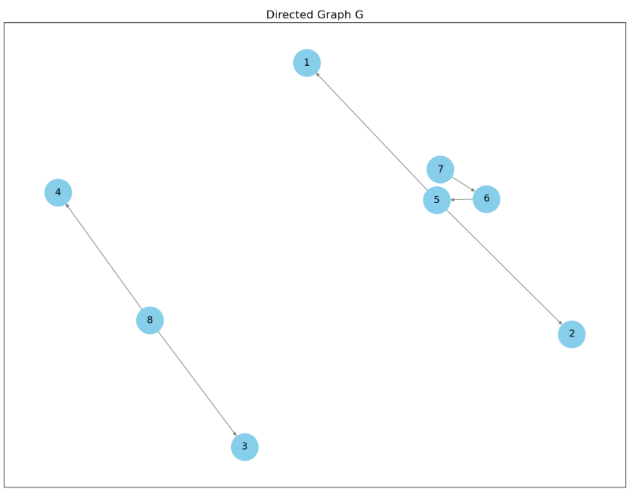
(a) *(7 points) Draw this directed graph using networkx package in Python and include the image.*

(b) *(3 points) What are the descendent and ancestor vertices of vertex 6?*

(c) *(2 points) How many connected components does the graph have?*

(d) *(3 points) Identify three pairs of vertices that form a cross edge (i.e., one is neither a descendent nor an ancestor of the other).*

a.



b.

Descendent Vertices Of Vertex 6 :7

Ancestor Vertices Of Vertex 6 :5

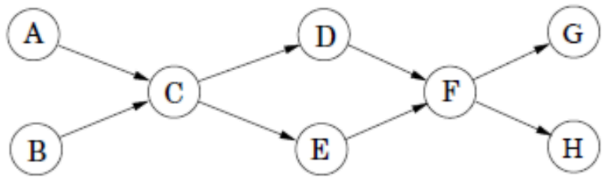
c.

CC is 2

d.

4/3; 1/2; 1/4

Question 2 (20 points). *Run the DFS-based topological ordering algorithm on the following graph. Whenever you have a choice of vertices to explore, always pick the one that appears first in alphabetical order.*



(a) *(12 points) Indicate the pre- and post- numbers of the nodes.*

(b) *(4 points) What are the sources and sinks of the graph?*

(c) *(2 points) Write down one topological ordering found by the algorithm.*

(d) *(2 points) How many topological orderings does this graph have?*

a.

Pre-order: {'A': 1, 'C': 2, 'D': 3, 'F': 4, 'G': 5, 'H': 7, 'E': 11, 'B': 15}

Post-order: {'G': 6, 'H': 8, 'F': 9, 'D': 10, 'E': 12, 'C': 13, 'A': 14, 'B': 16}

b.

Sources Of The Graph :A/B

Sinks Of The Graph: G/H

c.

BACEDFHG

d.

There is 1 topological ordering in this graph.

Question 3 (15 points). *Given an example of a graph with n vertices for which the queue of Breadth-first Search (BFS) will have $n-1$ vertices at one time, whereas the height of the recursion tree of Depth-First Search (DFS) is at most one. Both searches are started from the same vertex.*

BFS explores neighbors level by level while DFS follows a path as deep as possible before backtracking.

BFS start from node A, then the queue contains: {C, D, E, F, G, H} (size = n-1).

DFS also from node A, immediately visit all neighbors (size = 1)..

Question 4 (15 points). *You are given a binary tree $T = (V, E)$ that is not skewed and $|V| \geq 2$. Please describe in English a linear (in terms of $|V|$ and $|E|$) time algorithm to find the maximum sum of a path between any two leaves in T. Please explain why you are algorithm running time in linear.*

1.

The binary tree includes root, left tree and right tree. Due to the tree that is not skewed and |V | ≥ 2, computing the potential max path sum through this node:

maxSum = leftSum + rightSum + node.value

2.The reason for running time in linear:

Each node is visited only once → DFS traversal takes O(n).

Each edge is processed once → Every edge is used to compute the path sum, contributing to O(n).

the total runtime is O(|V|+|E|) that is O(n), which is optimal for tree traversal.

Question 5 (35 points). *For each vertex u in an undirected graph, let twodegree[u] be the sum of the degrees of u's neighbors. You are given an undirected graph $G = (V, E)$ in adjacency-list format.*

(a) (15 points) Please describe in English an efficient algorithm to compute the entire array of twodegree[·] values in time linear in $|V|$ and $|E|$.

(b) (15 points) Please write the pseudocode of your algorithm in (a).

(c) (5 points) Please explain why the running time of your pseudocode in (b) is linear in $|V|$ and $|E|$.

a.

Description:

1.Computing the degree of Vertex

- Traversal the adjacency-list. So store vertex and degrees of u's neighbor into array that is named by "degree".

2.Using two loop to traversal adjacency-list and sum the degrees of u's neighbor in the "degree".

Time Complexity:

Computing degrees traversals the vertex and edge: O(O(|V| + |E|)

Calculating twodegree --Each edge is visited twice → O(|E|) and traversal every vertex: O(O(|V| + |E|)

Overall complexity: O(|V| + |E|) (linear time)

b.

```
def c_two_degree(graph):
  degree = {}
  two_degree = {}
  for v, neighbors in graph.items():
    degree[v] = len(neighbors)

  for v, neighbors in graph.items():
    sum = 0
```

```
        for u in neighbors:
            sum += degree[u]
        two_degree[v] = sum

    return two_degree
```

c.

Step 1:

degree = {u: len(neighbors) for u, neighbors in graph.items()}

Computes the degree of each vertex in O(|V| + |E|) time.

Step 2:

For each vertex u, it sums the degrees of its neighbors: O(|V|).

Since every edge is counted twice in an undirected graph, it takes O(|E|).

Total Time Complexity:

O(|V| + |E|) — linear in the size of the graph.