Instructor Name: Siddhartha Nath
Email: s.nath@northeastern.edu

Northeastern University

Program Structure and Algorithms (INFO 6205)
Homework #3 –SAMPLE SOLUTIONS– 100 points

Student NAME:

Student ID:

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.* $\{1 : [4, 5], 2 : [7, 8], 3 : [12, 13], 4 : [14, 15], 5 : [3, 6], 6 : [2, 9], 7 : [1, 10], 8 : [11, 16]\}$.

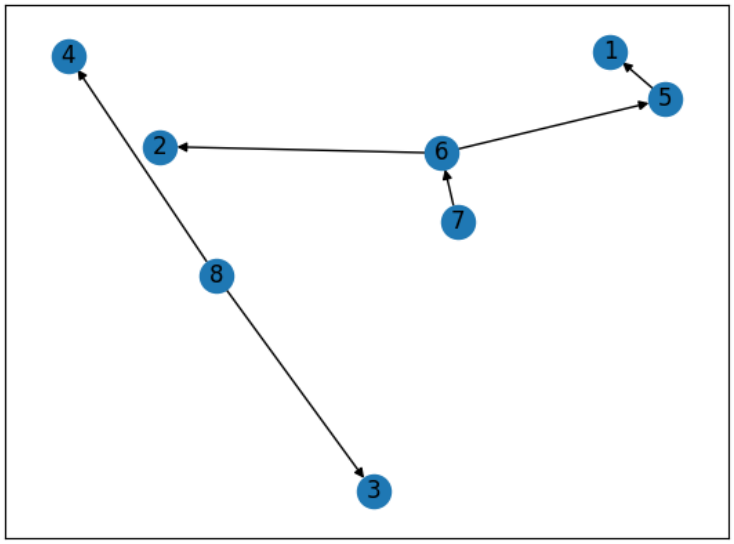
(a) (7 points) Draw this directed graph using networkx package in Python and include the image.

(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) Figure shows the (pre, post) numbers of each node.

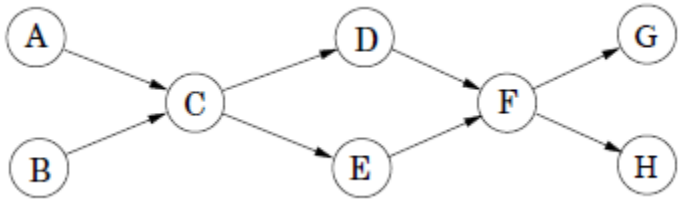


(b) Ancestor is 7, descendants are $1, 2, 5$.

(c) There are two connected components. $C_1 = \{1, 2, 5, 6, 7\}$ and $C_2 = \{3, 4, 8\}$.

(d) Three pairs: $(1, 2), (3, 4), (2, 5)$.

Question 2 (25 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.*



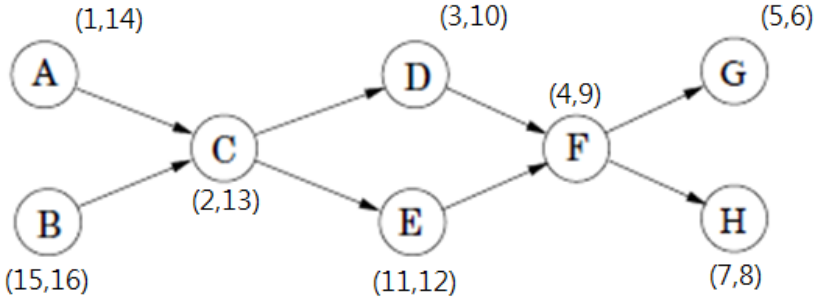
(a) (12 points) Indicate the pre- and post- numbers of the nodes.

(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) Figure shows the (pre, post) numbers of each node.



(b) The vertices A,B are sources and G,H are sinks.

(c) The algorithm outputs vertices in decreasing order of post numbers. One topological ordering found is B, A, C, E, D, F, H, G.

(d) Any ordering of the graph must be of the form $\{A, B\}, C, \{D, E\}, F, \{G, H\}$, where $\{A, B\}$ indicates A and B may be in any order within these two places. Hence the total number of orderings is $2^3 = 8$.

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.*

The following graph exhibits the property: $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$ and all edges are only from v_1 to each of $v_2, \ldots, v_n$, i.e., $E = \{(v_1, v_2), (v_1, v_3), \ldots, (v_1, v_n)\}$.

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.*

The maximum sum path between two leaf nodes may or may not pass through the tree's root node. We maintain a global variable for the maxSumPath. For every visited node x, we find the maximum root to leaf sum in the left and right subtrees of x. We add these two values with $x's$ value, and compare the sum with maxSumPath, if larger, we update maxSumPath. From each recursive call we return the sum of x and it's largest child.

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) *Algorithm description in English:*

- *First, we initialize all $|V|$ entries of the array degree[·] to 0.*
- *We calculate the degree of every vertex by counting the number of elements in its adjacency list. We do this by iterating through the adjacency list of every vertex and incrementing the value of the corresponding index of the array degree[·] by 1 in every iteration.*
- *Second, we initialize all the $|V|$ entries of array twodegree[·] to 0.*
- *Third, we iterate through the adjacency list of each vertex and increment the value of the corresponding entry of array twodegree[u] by the degree of each neighboring vertex of u.*
- *Finally, at the end of this, array twodegree[·] has the solution required.*

(b) *Pseudocode:*

```
   procedure calculateTwoDegree(Array adjacencyList[][])
1  for all u ∈ V:
2   degree[u] = 0
3  for all u ∈ V:
4   for all v ∈ adjacencyList[u]:
5    degree[u] = degree[u] + 1
```

```
6  for all u ∈ V:
7   twodegree[u] = 0
8  for all u ∈ V:
9   for all v ∈ adjacencyList[u]:
10    twodegree[u] = twodegree[u] + degree[v];
11  return twodegree
```

(c) ***Running time analysis:*** *We iterate through all the vertices twice and through all the edges twice. Therefore the running time of the algorithm* $= O(2 \cdot (|V|) + 2 \cdot O(|E|)) = O(|V| + |E|)$.