2201-COL106 Major

Anish Banerjee

TOTAL POINTS

34 / 35

QUESTION 1

9 pts

1.1 4 / 4

✓ + **1 pts** **|S|=1**

✓ + **1 pts** **Root is 1.**

✓ + **1 pts** **Left subtree correct**

✓ + **1 pts** **Right subtree correct**

+ **0 pts** **Incorrect**

1.2 5 / 5

✓ - **0 pts** **Fully Correct**

- **5 pts** Incorrect Explanation/Not Attempted

- **2 pts** The explanation includes why the given method will return x such that x is the ancestor of y, but doesn't include why it won't return any other node which is not ancestor of y. Just mentioned the fact that for x to be ancestor of y, x has to appear before y in preorder and after y in postorder.

- **1 pts** partially correct explanation: not clearly explained why the reasoning they have given is correct

QUESTION 2

2 5 / 5

✓ - **0 pts** **Correct**

- **5 pts** No answer

- **1 pts** Minor mistake

- **5 pts** Incorrect Algorithm

+ **1 pts** Correct idea but wrong implementation

+ **1 pts** Correct but inefficient algorithm

- **4 pts** Inefficient Algorithm

- **4 pts** Correct high-level idea; no implementation details.

- **4 pts** No implementation details

+ **1 pts** Vague idea.

QUESTION 3

3 4 / 4

✓ - **0 pts** **Correct**

- **1.5 pts** Minor errors

- **1 pts** Only correct value of node q returned

- **4 pts** Incorrect/Not attemepted

QUESTION 4

6 pts

4.1 2 / 2

✓ + **2 pts** **Correct**

+ **1 pts** Partially Correct / minor errors

+ **0 pts** Incorrect / Not attempted

4.2 4 / 4

✓ + **4 pts** **Correct**

+ **3 pts** Partially correct explanation / minor errors

+ **2 pts** Major errors (i.e., circular reasoning) / Provided an example but did not prove bound

+ **1 pts** Incomplete, but had some ideas right

+ **0 pts** Incorrect / Incomplete / Not attempted

QUESTION 5

11 pts

5.1 3 / 3

✓ - **0 pts** **Correct**

- **1 pts** 2.1 If condition incorrect

- **1 pts** 2.1 Else condition incorrect

- **1 pts** 2.2 Incorrect

- **3 pts** All incorrect

5.2 3 / 4

+ **1 pts** Topological sort complexity correct O(n+m)

✓ + **1 pts** **visit every vertex once O(n)**

✓ + **2 pts** **$$ \Sigma deg(v) $$ i.e**

outdegree/neighbours for all vertices O(m)

+ **0 pts** Not attempted or Completly incorrect solution

5.3 4 / 4

✓ **- 0 pts** **Correct**

- 4 pts Incorrect

- 4 pts Not attempted

COL106 Major Exam

Read the following instructions before you begin writing.

1. Keep a pen, your identity card, and optionally a water bottle with you. Keep everything else away from you, at the place specified by the invigilators. Switch off mobile phones before you put them away.
2. Write your entry number and name on every page. (Sheets will be separated prior to grading.)
3. Answer only in the designated space. Think before you use this space. No additional space will be provided for writing answers. Use the last blank sheet for rough work, if needed. Do not separate sheets from one another.
4. No clarifications will be given during the exams. If something is unclear or ambiguous, make reasonable assumptions and state them clearly. The instructors reserve the right to decide whether your assumptions were indeed reasonable.

Write your name in uppercase letters, one character per cell.

ANISH BANERJEE

Write your entry number, one character per cell.

2021CS10134

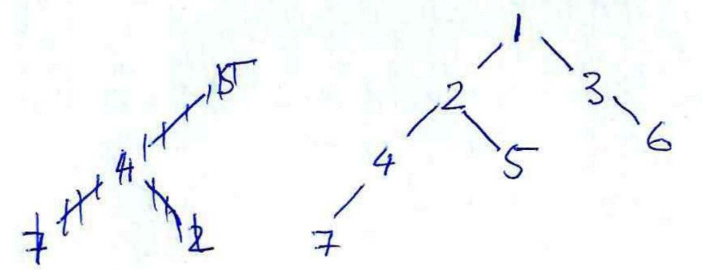
Name: ANISH BANERJEE Entry number: 2021CS10134

COL106 Major Exam Duration: 2 hours

1. **Binary trees**

   (a) (4 points) Let S be the set of binary trees (not necessarily search trees) over the set of keys $\{1, 2, 3, 4, 5, 6, 7\}$ that have in-order traversal $7, 4, 2, 5, 1, 3, 6$ and post-order traversal $7, 4, 5, 2, 6, 3, 1$. Answer the following questions.

   1. What is the size of S?
      Answer: 1
   2. Draw any one binary tree that belongs to S.



   (b) (5 points) Suppose you are given the pre-order and post-order traversals of a binary tree T over some set S of keys. Suppose $x, y \in S$. How will you determine whether x is an ancestor of y in T? Briefly explain why your answer is correct.

In post order traversal, first the keys of children will be printed and then the key of the node is printed. So, if y is a descendant of x then it must appear before x in the post order traversal. Similarly, in the pre-order traversal, x will occur before y if x is an anscestor of y

Post order: y before x $\iff$ y is descendent of x or y is in the left subtrees of x's anscestors ~~or y is ancestor of x~~

Pre Order: y after x $\iff$ y is the descendent of x or y is in the right subtree of x's anscestors

Intersection gives the anscestor relation

CHECK if y is before x in post order and after x in pre order

Name: ANISH BANERJEE Entry number: 2021CS10134

COL106 Major Exam Duration: 2 hours

2. (5 points) **String Processing**

Recall the pattern matching problem discussed in class and in the fourth assignment. Let Σ be a finite alphabet and x be a string over Σ. If p is a non-empty string over Σ, then we say that the pattern p matches x at index i if $x[i \ldots (i+m-1)] = p$, where m is the length of p. If p_1 and p_2 are non-empty strings over Σ, we say that the pattern $p_1?p_2$ matches x at index i if $x[i \ldots (i+m_1-1)] = p_1$ and $x[(i+m_1+1) \ldots (i+m_1+m_2)] = p_2$, where m_1 and m_2 are the lengths of p_1 and p_2 respectively. In other words, '?' is a wildcard character which is not in Σ, and which matches every character in Σ.

Suppose you are given a text x and two patterns, p_1 and p_2, which don't contain the wildcard. You have already computed L_1 and L_2, which are sorted lists of indices at which p_1 and p_2 respectively match x. Write down an algorithm to compute L, the list of indices at which the pattern $p_1?p_2$ matches x. Your algorithm must run in time linear in the total size of L_1 and L_2.

Let m_1 be the size of p_1 and m_2 be the size of p_2

$L \leftarrow [\ ]$; $i \leftarrow 0$; $j \leftarrow 0$

while $i < m_1 - 1$ and $j < m_2 - 1$:

if $L_1[i] = L_2[j] - m_1 - 1$ then

L.append($L_1[i]$)

$i = i+1$; $j = j+1$

else if $L_2[j] < L_1[i] + m_1 + 1$:

$j = j+1$

else if $L_2[j] > L_1[i] + m_1 + 1$:

$i = i+1$

end if;

end while;

return L

Name: ANISH BANERJEE Entry number: 2021CS10134

COL106 Major Exam Duration: 2 hours

3. (4 points) **(2,4)-trees**
Recall the procedure for deleting a key from a $(2,4)$-tree: if we find the required key in a non-leaf node, then we exchange it with its successor and then delete it. But how do we find its successor in the first place? Your job is to write a `python` code to find the successor of a key in a $(2,4)$-tree that resides in a non-leaf node. A node in our $(2,4)$-tree has the following attributes.

- `key`: a list of 3 objects. If the node contains k keys (where $1 \leq k \leq 3$), then these keys are `key[0]`, ..., `key[k-1]`, while `key[k]`, ..., `key[2]` are all `None`.
- `child`: a list of 4 node-references. If the node has d children (where $0 \leq d \leq 4$), then the references of these children are `child[0]`, ..., `child[d-1]`, while `child[d]`, ..., `child[3]` all `None`.
- `parent`: this is `None` if the node is the root of the tree, else the reference to the parent node.

Complete the following function which takes a reference `p` of a non-leaf node and an integer `i` as input parameters, and returns a pair `(q,j)`, where `q` is a node-reference and `j` is an integer such that `q.key[j]` is the successor of `p.key[i]` in the $(2,4)$-tree. The function must run in time $O(\log n)$, where n is the number of keys in the $(2,4)$-tree.

```
def findSuccessor(p, i):
    if p.key[i] is None: return
    q = p.child[i+1]
    while q.child[0] is not None:
            q = q.child[0]
    j = 0

    return (q, j)
```

Name: ANISH BANERJEE Entry number: 2021CS10134

COL106 Major Exam Duration: 2 hours

4. **Undirected Graphs**
Recall that we defined the distance between two vertices of an undirected graph to be the length of a shortest path between those vertices (for example, the distance between a vertex and itself is 0, the distance between adjacent vertices is 1, and so on). The *diameter* of a graph is defined as the maximum, over all pairs u, v of its vertices, of the distance between u and v.

Let G be a connected undirected graph. Perform a breadth-first-traversal of G starting from some vertex s, and let T be the resulting breadth-first-traversal-tree. Suppose T has exactly $h+1$ levels $L_0 \ldots, L_h$, where $L_0 = \{s\}$.

(a) (2 points) Prove that the diameter of G is at least h.

Consider a vertex v at the last level of the BFT Tree.
By ~~p~~ appending to a list ~~prev~~ v and prev[w] for all ancestors
w of v, we obtain a list which has a path of
length h (from v to s). The diameter of the
graph must be $\geq$ this path length.
So $\text{diam}(G) \geq h$

(b) (4 points) Prove that the diameter of G is at most $2h$.

Consider two vertices v_1, v_2 in the last level L_h of
the BFT Tree. In part (a) it is proved that there
exists a path of length h between (s, v_1) and (s, v_2)
So $\quad \text{dist}(s, v_1) \leq h \quad \text{dist}(s, v_2) \leq h$ — (*)
(This is the max possible distance from s)
By Triangle inequality of graphs
$\text{dist}(v_1, v_2) \leq \text{dist}(s, v_1) + \text{dist}(s, v_2) \leq 2h$ — (†)
If v_1, v_2 were not on the last level, then the distance
between them and s would be $< h$. (*) still holds.
So (†) holds for any vertices $v_i, v_j \in V$. Hence $\text{diam}(G) \leq 2h$

Name: ANISH BANERJEE Entry number: 2021CS10134

COL106 Major Exam Duration: 2 hours

5. **Directed Acyclic Graphs**
Recall the out-adjacency list representation of directed graphs, where we have one linked list for every vertex v, where the linked list contains the out-neighbors of v. Given a directed acyclic graph G in its out-adjacency list representation and a vertex t of G, we would like to compute, for every vertex v of G, the number of directed paths from v to t. Note that the number such paths could be exponential in the number of vertices, so a brute-force counting is too inefficient.

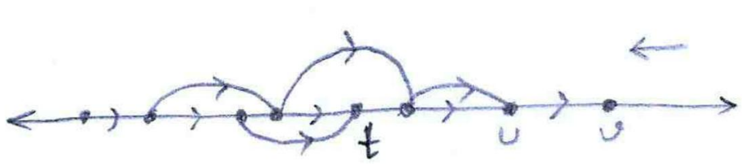
(a) (3 points) Complete the following algorithm to compute an array `pathCount` indexed by the vertex set of G so that at the end of the run of the algorithm, `pathCount[v]` equals the number of directed paths from v to t in G. Each blank must be filled up with an expression that can be computed in $O(1)$ time.

`findPathCounts(G,t)`

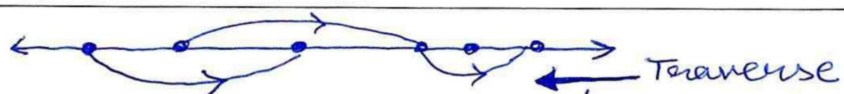
1. Compute a topological sort of `G`.
2. For each vertex `v` in the reverse of the topological sort order:
   2.1. If `v=t` then `pathCount[v]` ← 1, else `pathCount[v]` ← 0.
   2.2. For each out-neighbor `u` of `v`: `pathCount[v]` ← pathCount[v]+path Count [U]
3. Return `pathCount`.

(b) (4 points) Prove that the above algorithm runs in time $O(n+m)$, where n is the number of vertices and m is the number of edges in G.

The for loop in step 2 traverses through all the vertices in the Graph. — $O(n)$
For each vertex, we are checking all its out neighbours. So $\sum_{v \in V} d_{out}(v) = |E| = m$
Hence net complexity becomes $O(n+m)$



(c) (4 points) Why is it necessary to iterate over the vertex set in reverse topological sort order in the algorithm of part (a)? Explain briefly.

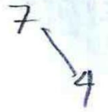


Recall that in a topological sorting, the edges can go only from left to right. If we iterated in the topological order, then in Step 2.2, we would end up at out neighbours that haven't been visited yet and our algorithm fails.
In reverse Top. sort., we ensure that all the out neighbours have been visited before.

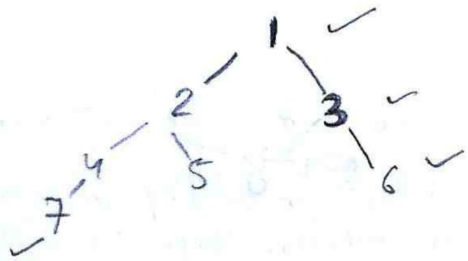
In 7 4 2 5 1 3 6

Post 7 4 5 2 6 3 1

Pre 1 2 4 7 5 3 6



[child
me

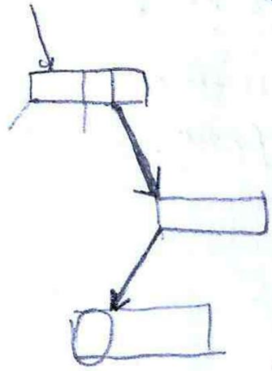


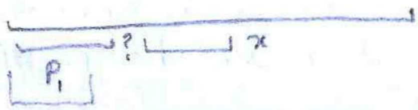
7

$[i_1, i_2 \ldots\ldots i_k]$

[child
me

me
[child





post
y before x iff
y is descendant of
x or y is in the
left ~~right~~ subtrees of
x's anscestors

```
a = p.child[i+1]
while a.child[0] is not None:
    a <= a.child[0]
q = a
```

Pre: y after x iff
y is descendant of x
or y is in the right
subtree of x's anscestors