CS180-Fall20 Midterm-10am

[redacted]

TOTAL POINTS

94 / 100

QUESTION 1

1 Q1 **18 / 20**

- **- 0 pts** Correct
- **- 2 pts** BFS may not find all nodes
- ✓ **- 2 pts No need to start BFS from a node without incoming edge in DAG**
- **- 4 pts** Proof of correctness is not complete or partially correct
- **- 8 pts** Proof of correctness is wrong/missing
- **- 2 pts** Time complexity analysis is partially correct
- **- 4 pts** Time complexity analysis is wrong/missing
- **- 8 pts** BFS search is wrong

QUESTION 2

2 Q2 **20 / 20**

- ✓ **- 0 pts Correct**
- **- 10 pts** incorrect algorithm
- **- 5 pts** incorrect time complexity
- **- 5 pts** incorrect proof
- **- 3 pts** unclear algorithm statement
- **- 2 pts** time complexity can be more precise
- **- 2 pts** informal/incomplete proof

QUESTION 3

3 Q3 **16 / 20**

- **- 8 pts** Incorrect/No algorithm
- **- 4 pts** Unclear/partially correct algorithm
- **- 8 pts** Incorrect/No proof
- ✓ **- 4 pts Unclear/Informal/Incomplete proof**
- **- 4 pts** Incorrect/Not optimal time complexity
- **- 0 pts** Correct

QUESTION 4

4 Q4 **20 / 20**

- ✓ **- 0 pts Correct**
- **- 5 pts** Incorrect stack operations (if stacks drawn) or incorrect edges traversed (if trees drawn)
- **- 10 pts** Vertices explored in incorrect order
- **- 10 pts** Step-by-step not shown, but some reasoning given
- **- 10 pts** Missing or fully incorrect steps
- **- 2 pts** Vertices in final DFS tree not added via discovery
- **- 2 pts** Unnecessary edges in final DFS tree
- **- 5 pts** Missing or fully incorrect DFS tree

QUESTION 5

5 Q5 **20 / 20**

- **- 10 pts** Incorrect Algorithm or did not attempt algorithm
- **- 3 pts** No discussion of correctness
- **- 3 pts** No discussion of runtime
- **- 20 pts** Did not attempt
- **- 5 pts** Approach attempted but insufficient.
- ✓ **- 0 pts Correct**
- **- 3 pts** incorrect runtime analysis

Name(last, first) [redacted]

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** **ID :** [redacted]

Midterm **Total Time: 1.5 hours** **November 2, 2020**

Each problem has 20 points .

All algorithms should be described in bullet format (with justification/proof). You cannot quote any time complexity proofs we have done in class: you need to prove it ~~yourself~~. Assuming we want to traverse?

Problem 1: Describe the Breadth First Search algorithm in a DAG. Prove its correctness. Analyze its complexity.

Find the first node with indegree 0. (or some other chosen root).
Begin with all nodes unvisited, and an empty queue.

Push our root node onto the queue.

While the queue is not empty:

Pop the first node off the queue.

Denote this node as n.

Add it to our tree

For all neighbors of n:
if the neighbor has not been visited,
AND there is a directed edge n → neighbor,

add it to the queue.

Time:
We visit each node once, and perform a constant number of operations per node, We visit each edge a constant number of times, therefore, overall complexity is O(m+n).

Proof:

Our BFS will traverse all connected & accessible nodes:

If all nodes are connected, but our BFS has terminated without visiting the node:

The node must have been pushed onto the queue and not processed (which means BFS has not finished), a contradiction.

If the node was not pushed onto the queue, there must have been no edges directed towards it (in-degree 0) which means it is inaccessible.

1 Q1 18 / 20

- **- 0 pts** Correct
- **- 2 pts** BFS may not find all nodes
- ✓ **- 2 pts** **No need to start BFS from a node without incoming edge in DAG**
- **- 4 pts** Proof of correctness is not complete or partially correct
- **- 8 pts** Proof of correctness is wrong/missing
- **- 2 pts** Time complexity analysis is partially correct
- **- 4 pts** Time complexity analysis is wrong/missing
- **- 8 pts** BFS search is wrong

Name(last, first): ___________________

Problem 2: Consider a set of intervals/tasks. Each task has a start and an end time and each processor can handle one task at any given time. If tasks do not overlap, then we can use one processor to schedule them all. If they do overlap, we need more processors to schedule them. For example, in the figure below we need two processors to schedule all four intervals/tasks. A. Design an algorithm that finds the minimum number of processors needed to schedule all intervals/tasks. B. Analyze the time complexity of your algorithm. C. Prove the correctness of your algorithm.

Processor 1 ---------------------- ----------------------------
Processor 2 ------------------------------ ----------------------------

Greedy Alg:

Given set S of tasks, list P of processors

- Initialize P with 1 processor
  - Each processor keeps track of its end-time.
    - A processor is "free" if the task we are trying to assign has start time > end time.
    - When we assign a task to a processor, it inherits the task's end time.
- Sort the tasks by start time.
- As long as there are tasks remaining:
  Choose the first remaining task T, and add it to the first free processor.
  - For all tasks overlapping with T: (some set t')
    - Choose the first task in t'
      - Assign it to the first avail. processor
      - If there are none free, add a new one
- Return length of processor list.

Time Complexity:

- Sorting: $O(n \log n)$ (known)
- Assigning tasks: Worst case n processors, n^2 operations

Total: $O(n^2)$

Average case assuming # processors << n:

$O(n \log n)$

Proof:

If a processor is added, then all other processors must be busy / overlapping. Therefore, that processor is needed to accomodate all tasks.

2 Q2 20 / 20

✓ - 0 pts Correct

- 10 pts incorrect algorithm
- 5 pts incorrect time complexity
- 5 pts incorrect proof
- 3 pts unclear algorithm statement
- 2 pts time complexity can be more precise
- 2 pts informal/incomplete proof

Name(last, first): ___________________

Problem 3: Design an algorithm that decides if a connected undirected graph is 2-colorable and finds a 2-coloring if it is indeed 2-colorable. Prove the correctness of your algorithms and analyzes its time complexity.

For simplicity, assume we will attempt to 2-color with R(red) and B(blue).

Algorithm:

Color our start node B(blue) and push it to the queue
Run BFS with the following modifications:

- When we are examining neighbors of a node to push onto the queue, color the neighbors the opposite color of our current node
  - If a node we encounter has already been colored a different color, exit - our graph IS NOT 2-colorable
  - otherwise, continue BFS as usual.

If we finish BFS successfully, output the colored graph and decide that our graph IS 2-colorable see #1

Time complexity:

BFS is known to be O(m+n). Our modifications to BFS involve a constant number of color comparisons per edge and a max of n colorings of nodes, so the modified algorithm is still O(m+n).

Proof:

If the graph IS 2-colorable and we determined it was not:

- If it is 2-colorable, then nodes in any layer must only have edges to layers immediately above or below them, not within their own layer.
- Our algorithm alternates colors layer by layer
- If the above holds true, then we cannot have determined the graph is not 2-colorable, since we only determine that if a node is a neighbor to a node in its own layer or layers
- Therefore, our algorithm must have exited and concluded YES a contradiction

If a graph is NOT 2-colorable:

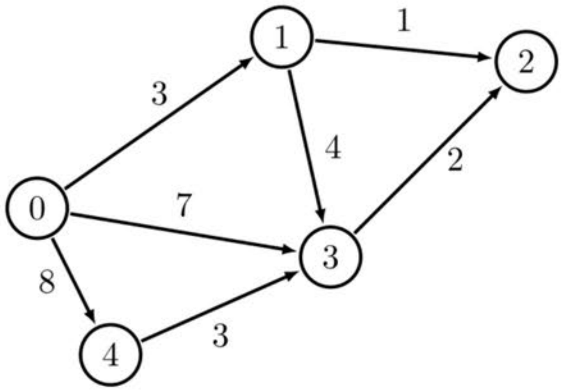
- There must be some odd-number cycle
- Our BFS will encounter the same node on the cycle twice, from 2 origin nodes of different colors, meaning it must have exited and concluded the graph is NOT 2-colorable.

3 Q3 **16 / 20**

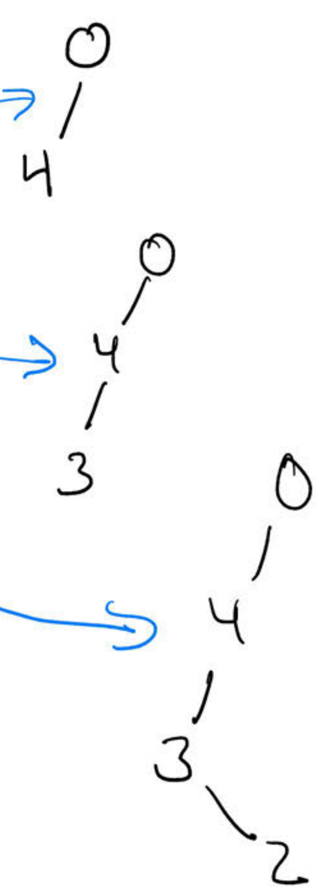
- **- 8 pts** Incorrect/No algorithm
- **- 4 pts** Unclear/partially correct algorithm
- **- 8 pts** Incorrect/No proof
- ✓ **- 4 pts Unclear/informal/incomplete proof**
- **- 4 pts** Incorrect/Not optimal time complexity
- **- 0 pts** Correct

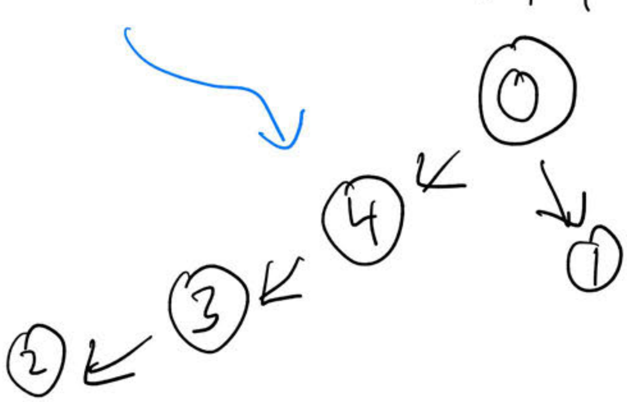
Name(last, first): ___________________

Problem 4: Apply the DFS algorithm to the graph shown below (step by step) starting from vertex zero (0), and show the final DFS Tree. (You can ignore the weight on the edges.)



Step	At	Stack	Visited	Tree
1	0	0,		0,
2		4,3,1	0	
3	4	3,1	0,4	
4		,3,1	0, 4	
5	3	2, 1	0,4,3	
6	2	1	0,4,3,2	
7	1		0,4,3,2,1	





4 Q4 **20 / 20**

✓ **- 0 pts** Correct

- 5 pts Incorrect stack operations (if stacks drawn) or incorrect edges traversed (if trees drawn)

- 10 pts Vertices explored in incorrect order

- 10 pts Step-by-step not shown, but some reasoning given

- 10 pts Missing or fully incorrect steps

- 2 pts Vertices in final DFS tree not added via discovery

- 2 pts Unnecessary edges in final DFS tree

- 5 pts Missing or fully incorrect DFS tree

Name(last, first): ___________________

Problem 5: It takes n-1 comparisons to find the minimum number in a given list of integers L = (x1, x2, x3, ….). Similarly, it takes n-1 comparisons to find the maximum. Therefore, it is trivial to design an algorithm that finds both the minimum and the maximum with about 2n-2 comparisons. Design an algorithm to find both the minimum and the maximum in a list using about 3n/2 comparisons.

Given a list of n unsorted numbers L:

$a \Leftarrow 1$

$b \Leftarrow n$

While $a \neq b$:

If $a < b$:

Add a into a list X of smaller numbers

Add b into a list Y of bigger numbers

If $a > b$:

Add a into a list Y of bigger numbers

Add b into a list X of smaller numbers

If a = b, or there is 1 element remaining:

Add a into both lists

Find the max of Y:

Trivial - keep track of the largest element so far, iterate and compare

Find the min of X:

Trival

Time Complexity

- $(\frac{n}{2})$ comparisons to split the list into X & Y, which have $(\frac{n}{2})$ elements each
- $(\frac{n}{2}-1)$ comparisons to find the max of Y, $(\frac{n}{2}-1)$ to find the min of X

5) continued

Total complexity: $\left(\frac{n}{2}\right) + 2\left(\frac{n}{2} - 1\right) = \frac{3n}{2} - 2$

$= O\left(\frac{3n}{2}\right)$ as desired.

Proof:

1) Algo, finds the correct min

- If the actual min ends up in X, the algorithm will find it. (simple min, by comparisons)
- the actual min must be in X.
  - If the min is compared with any other number, it will be lesser and be added to X, except if:
  - If the number is added to Y because there were equal integers or an odd # of ints, it would have also been added to X.

2) We will find the correct max

- Same logic as min, switch X ⟺ Y and less ⟺ more etc.

5 Q5 **20 / 20**

- **- 10 pts** Incorrect Algorithm or did not attempt algorithm
- **- 3 pts** No discussion of correctness
- **- 3 pts** No discussion of runtime
- **- 20 pts** Did not attempt
- **- 5 pts** Approach attempted but insufficient.
- ✓ **- 0 pts** Correct
- **- 3 pts** incorrect runtime analysis