87/100

Name(last, first): [redacted]

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** **ID (4 digit):** [redacted]

Midterm **Total Time: 1.5 hours** **February 2016**

Each problem has 25 points.

20

1 **A.** Describe Topological Sort (in English, bullet by bullet) on a DAG **B.** Analyze its time complexity and justify your answer. **C.** What will happen when there is a cycle? Prove your answer.

A.
- Count the number of incoming edges for every node. and label them so.
- Find a source (a vertex with no incoming edges), and put it in the sorted list. Remove the vertex from the graph by decrementing the #s of incoming edges for all of its neighbors.
- Repeat the step above until all vertices have been removed.
- The sorted list of vertizes is a topological sort.

B The time complexity is $O(m+n)$. This is because you remove n vertices and m edges over the course of the sort. (there is also an additional linear time to count the number of incoming edges per node, but that is included in $O(m+n)$).

(-5) step 2 takes $O(V)$ for V runs $\rightarrow O(V^2)$

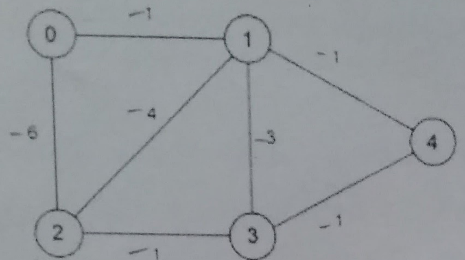
C If there is a cycle (which by definition DAGs do not have), this means there is no source, and thus the topological sort has nowhere to begin.

Proof: Assume there is a graph with no sources, show that there is no cycle. If every vertex in the graph is not a source, that means they each have at least one incoming edge. Start with a vertex x, which connects to y, which connects to z, etc. Eventually there will be a vertex u that must connect to x. Therefore there cannot be a source in a graph with a cycle, and thus the topological sort does not work.

24

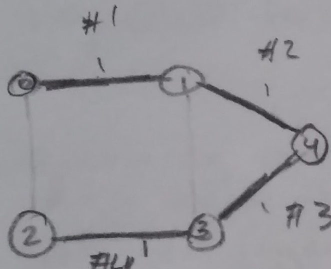
Name(last, first): [redacted]

2. A. Use Kruskal's MST algorithm to find a an MST in this graph. Show each step.
B. If some edges were negative would the algorithm still find an MST. Prove your answer.



A. ① Sort edges: 1, 1, 1, 1, 3, 4, 6

② Place smallest edge suchthat it doesn't Form a cycle: (steps shown with '#'). Repeat untill all vertices are in MST.



← All vertices are in MST
Weight = 4

B. Yes, if some of the edges were negative the algorithm would still work. This is because Kruskal's always chooses the smallest edges possible (i.e. ones that don't form a cycle). Negative edges will be smaller than the positive ones. The only difference then is that the weight may be negative.

- close, it's the relative order

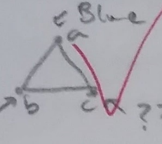
Name(last, first): [redacted]

25

3. A graph is two-colorable if we can color its vertices with RED & BLUE such that no two adjacent vertices have the same color. **A.** Are all graphs two-colorable. Prove your answer **B.** Design an efficient algorithm for two coloring a graph. Prove the correctness of your algorithm. **C.** Analyze its time complexity. **D.** How many colors do you need to color a tree?

A. No, not all graphs are two-colorable. Graphs which are bipartite are not two-colorable because they contain an odd cycles and you can't color odd cycles with two colors.

Proof: Simple counterexample →



this is a graph that is not two-colorable. Vertex c cannot be colored blue or red because it is adjacent to two vertices that are blue and red.

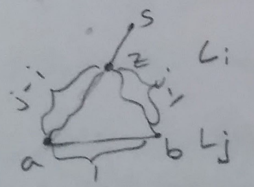
B. • Arbitrarily choose a vertex of the graph and run a BFS from it

• On odd layers (L_i), color the vertices blue. On even layers, color vertices red.

• Run a check to see whether every edge touches two different colored vertices. If not, return False (not two-colorable). Otherwise, return the coloring.

Proof

My algorithm colors vertices that are in the same layer the same color. That means that if there are two vertices that are the same color with an edge between them, they belong to a graph that is not two-colorable (i.e. a non-bipartite graph). It is proven below:



Say a and b are in the same layer L_j and have an edge between them. This means, if they are in the same layer, they share some ancestor z in some layer L_i

To calculate the length of the cycle between z, a, and b, simply add the distances: $(j-i)+1+(j-i) = 2(j-i)+1$. This is an odd number, thus it is an odd cycle, and thus the graph is not bipartite.

Therefore my algorithm is correct

continued on other side 3 ↘

C. Time complexity: O(m+n).
the BFS takes m+n steps. and there is another m+n
check afterwards, so O(m+n).

D A tree by definition has no odd cycles as it has
no cycles at all, so it is two-colorable.

0	0	5	12	9	13	9

max: ~~0~~ ~~8~~ ~~12~~ 13

Name(last, first): [redacted]

18/25

4. Consider a sequence x1, x2, ... , xn of (positive and negative) integers. We want to find two indices i and j such that xi + ... + xj is maximized.
A. Describe in English (bullet by bullet) an O(n) time algorithm (using constant extra space) for solving this problem. For example, if the input is (-2, -2, 5, 7, -3, 4, -4) then i=3 and j = 6 (and the sum xi+…+xj is equal to 13). **B.** Prove the correctness of your algorithm.

- Create a max sum (max) and set it to 0

15 A.
- Create an n-length array called "current sum" that holds the current sum for every integer (all initialized to 0). uses O(n) extra space -2

(Starting at the lefthand side of the sequence),
- Calculate and set the current sum, which is the current integer plus the current sum of the previous element. If this result is <0, set that element's current sum to be 0 instead.
- If the current sum is greater than or equal to the max sum, set max to the current sum.
- Repeat the two previous bullets until all integers are processed
- Going from the end of the list, go backwards until you hit the integer whose current sum equals the max sum. the position of that integer is j.
- Continue going to the left (towards the beginning of the list) and stop when you hit an integer whose current sum is 0 or stop when you get to integer at the beginning of the list The position of that integer is i.
- Return i and j

3 B. Proof that the algorithm at any given step, the max sum contains the maximum sequence at that point.
Induction (with ~~an~~ optimal sequence)
BASE CASE: 1 integer in a sequence, sequence is same for algorithm and optimal.

INDUCTIVE STEP: Assume our algorithm finds the maximum sum for n-1 integers. Show that it does for n integers as well

continued on other side

Say adding x_n to max will increase max.

$\underbrace{x_1, x_2, \dots x_{n-1},}_{\text{maximum sum is same as optimals:}} X_n \dots$

algorithm's max = max   optimal $\overset{\text{largest}}{\text{sequence}}$ max = max'

Our algorithm will only update the max sum if the current sum of $x_{n-1} + X_n \geq$ max sum
The only way max' > max is if our algorithm chooses not to update the max, which is impossible by the definition, or if the current
? sum of $x_{n-1} + x_n <$ max sum which is also impossible because we defined x_n to be part of the optimal $\overset{\text{largest}}{\text{sequence}}$.
This proves that because my algorithm always updates the max sum if an element will increase it, and thus it always finds the max sequence,
and so my algorithm is correct.