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.

24

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)
- Find the degree of each vertex
  - consider each edge & add 1 to receiving vertex' degrees. //O(e)
- Create a queue to hold source nodes.
- Add a source node to this queue //O(V)
  - if one does not exist, return that there is no topological sort
- While there are sources in the queue
  - pop off a source, node s & add to topological sort
  - remove this node s from the DAG & any edges it has //O(e)
    - Update the vertexes these edges point to (subtract 1)
    - if any vertexes became degree 0, add it to the source queue

endwhile

- if DAG is empty
  return the topological sort
- else
  return no topological sort.

b) Finding vertex degree is O(e)+V -1 b/c you must consider each edge. You will add at most V nodes into the source node queue which is O(V). You will remove e edges & v nodes: O(e+v). O(2e+2v) = **O(e+V)**

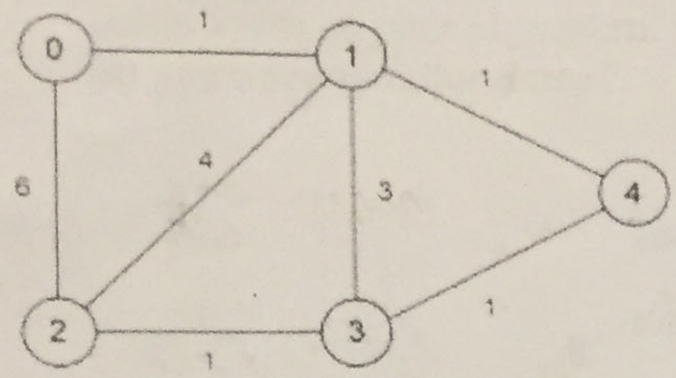
c). C is on back of this page

c) If there is a cycle, none of these vertexes will become source nodes. For one of these to become a source node, one must be removed, but this is impossible because we only remove source nodes. Thus the while loop will end before any of these nodes in the cycle are removed and the algorithm will return that there is no topological sort. as the DAG is not empty.

Name(last, first): [redacted]

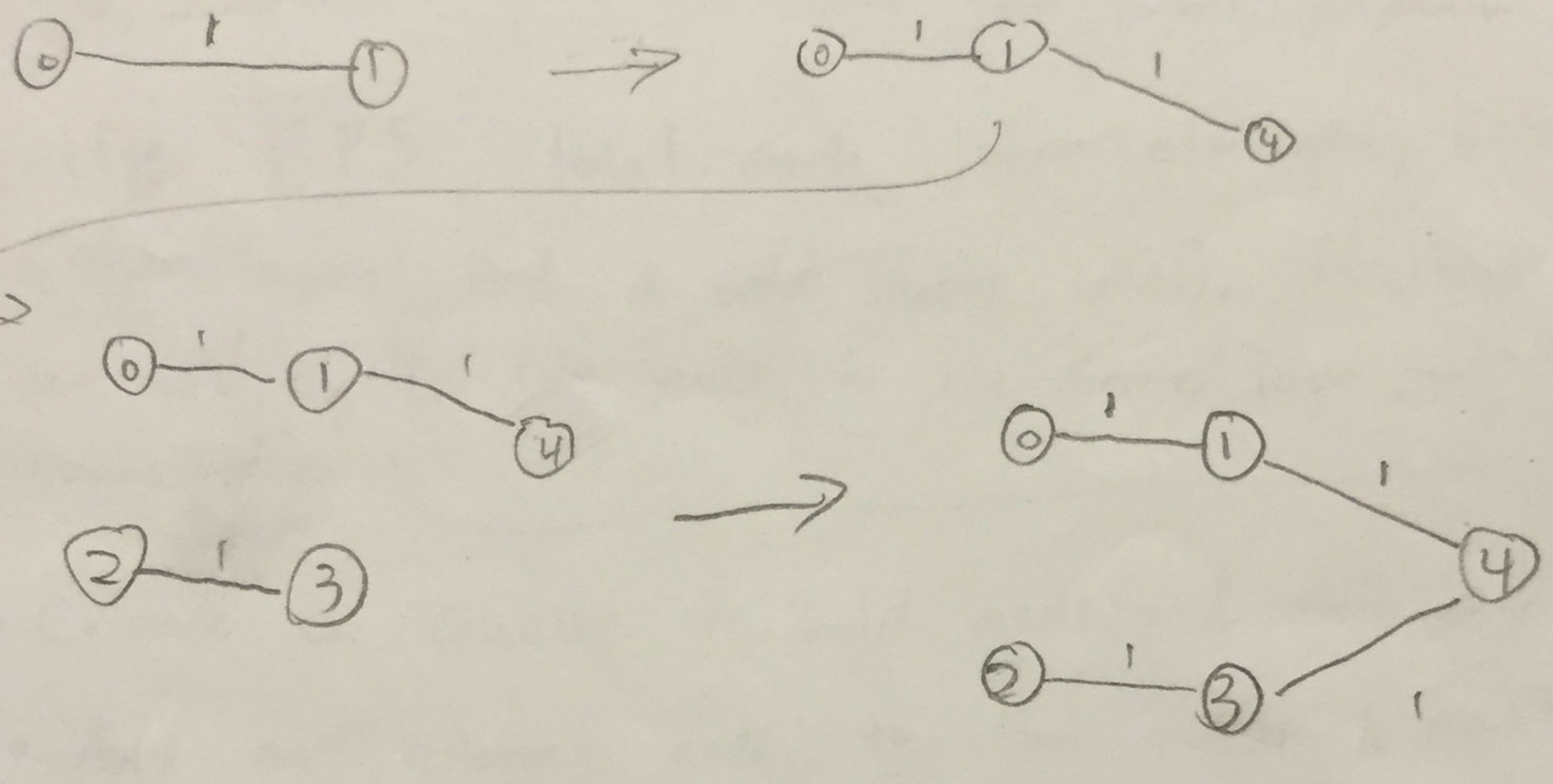
23

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.



I will
★ arbitrary consider edges on 0 first, then 1, then 2, then 3, then 4, then select the first minimum found.
cycle check?

A)



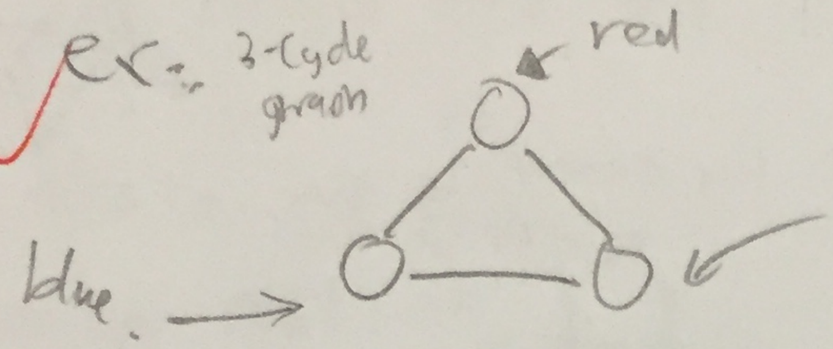
B). Yes, Kruskal would add all the negative weights first. Here's my logic. if the lowest neg number is X, then if you add $|X| + 1$ to every number, (no negs) Kruskal would work, so why would subtracting a set amount from each edge affect anything. You would add the edges in the same order. Your total would be the same as the positive instance only subtract $V(|x| + 1)$, but you subtract that from every possible spanning tree, so it will still be the minimum spanning tree. (of the positive instance)

20

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. if we have an odd cycle it will not work.

ex. 3-cycle graph



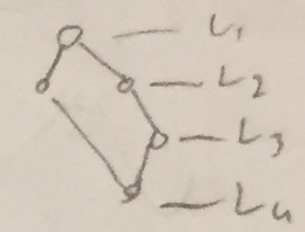
can't color this one w/o have two same colors adjacent.

B). Use BFS, label each layer alternating colors. (even layers red & odd layers blue). If there is an odd cycle, two nodes in the same layer will be connected.

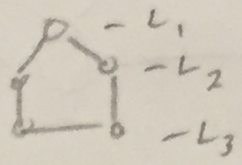
- Create a queue to hold nodes. & their layer.
- Add an arbitrary node to the queue. & mark it layer 0.
- while the queue is not empty
  - Pop off a node s w/ layer i
  - mark s as visited w/ layer i & mark it blue if i is even & red otherwise
  - travel along every edge & check if that vertex visited.
    - if not, add it to the queue w/ layer i+1
    - if so, if it's layer is i, ☆ or if the same color. Either check will work.
      return that there is no way to two color the graph
- endwhile.
- return the coloring

Proof (C & D) on the back of this page.

B) Proof: If there is an odd cycle, two nodes will be in the same layer. If there is an odd cyle it must travel from layer i to layer j & back. This distance will be $(j-i)2$, which is even. To make it odd, the nodes will connect becoming distance $2(j-i)+1$. But why can't



happen? The node in L_4 cannot be there, it must be in L_3...



like so, which proves my point

Using this idea, my algorithm checks if a node travels to another visited node in the same layer. If this occurs, there must be an odd cycle. Each of these nodes are i away from the source & 1 from each other $(2i+1)$. Since there will always be two nodes in the same layer that are connected, my algorithm will always find the odd cycle if there is one...

See part A for why an odd cycle does not let you two color

C). • You add at most V nodes to the queue, $O(V)$,
• While checking, you travel along all e edges $O(e)$

Total: $\boxed{O(e+V)}$

D). If there are n vertices in the graph, then $\boxed{n \text{ colors}}$

~~Could be needed if the entire graph is connected.~~

Each vertex would need its own color as it is adjacent to all $n-1$ other vertices.

Name(last, first) [redacted]

24/25

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.

17 A)

- Sum = 0; max = 0; i = 1; imax = 0, jmax = 0
- For k = 1, 2, 3, ... n // O(n)
  - sum += xk;
  - if sum < 0
    - sum = 0
    - i = k+1 // to point to next value.
  - if sum > 0
    - if sum > max
      - max = sum;
      - imax = i;
      - jmax = k;

End for

- return max, imax & jmax // max is 0 if all neg., so is imax & jmax if all neg.

```
if (max == 0) // all negative
for k = 1, 2, 3, ... n // O(n)
    sum = x k;
    if sum > max
        imax = sum
        imax = k
        jmax = k
end for
```

7 B) the idea behind this is if your current sum is > 0, you want to keep it because it can only increase the max. if the sum is < 0, it will decrease your max, so better to just dump it & start a new. To do this we set our sum to 0 & have our starting index reset. if the sum is positive, we want to keep it. If we reached a new maximum, we want to log our i & the current position we are at as the endpoint

~~My initial loop will fail if we are passed only negative integers... (It said it could pass both pos & neg but just to be safe I added another loop. that is O(1), so this would take O(2n) = O(n) for all neg integers.)~~

more on back...

B) Proof.

Assume there is a sequence of higher value than the one I found

Case: my sequence is included in this sequence

- There must be a positive int either to the left or right
- If this is the case it would have been included

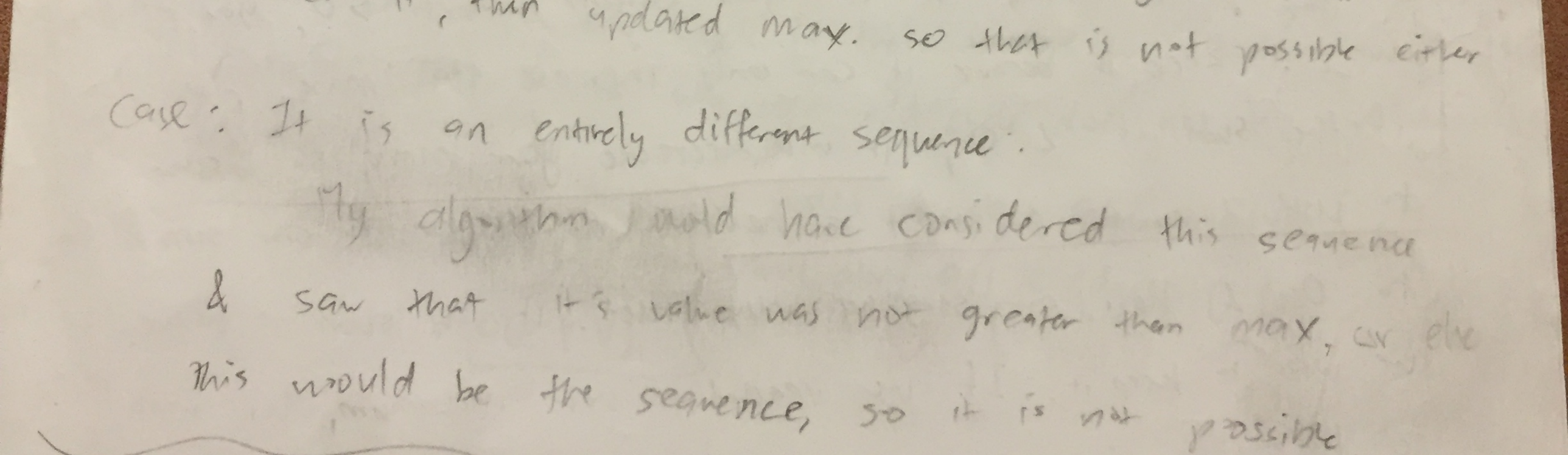
The previous things before mysequence either added to neg, or it would of included it. So if we had -100, 5, my sequence, the -100 would of been ignored, sum set to 0, 5 would be added & then my sequence would have included it, so it's impossible. Also note the max sequence would have to start w/ a positive unless all values are negative.

If a positive number came after mysequence my algorithm would have added it, then updated max. so that is not possible either

Case: It is an entirely different sequence.

My algorithm would have considered this sequence & saw that it's value was not greater than max, or else this would be the sequence, so it is not possible.

The main way my algorithm would fail is if my neg sum idea failed. That is a sequence has the beginning sum to neg, then add to max. This would be the case:



Better to just drop the neg total & reset sum & starting index