Name(last, first): Peterman, Ryan

89/100

# UCLA Computer Science Department

**CS 180** **Algorithms & Complexity** **ID (4 digit):** 9922

**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) Topological Sort:

First we define a source as a node with in-deg = 0

1. Calc indegree and outdegree of all nodes iterating over edges
2. Generate a list of sources by checking all nodes
3. pick/output an arbitrary source from source list and remove it
4. remove source from graph and update adjacent node degrees
5. if any nodes became sources add them to source list
6. repeat 3,4,5 until all sources picked/output

b) $O(V+E)$ because the following analysis of each step:

9

1. $O(E)$ since iterating over edges
2. $O(V)$ since iterating over vertices

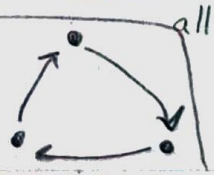
loop $O(V)$:

3. $O(1)$ to pick arbitrary source and remove it
4. $O(1)$ to remove node from graph and update
5. $O(1)$ to check if a node became a source while updating
6. $O(V)$ loop until all verts output

$\therefore$ Thus, $O(E+V)$

c) If there is a cycle, the algorithm will terminate early since there are no sources in a cycle.

ie.



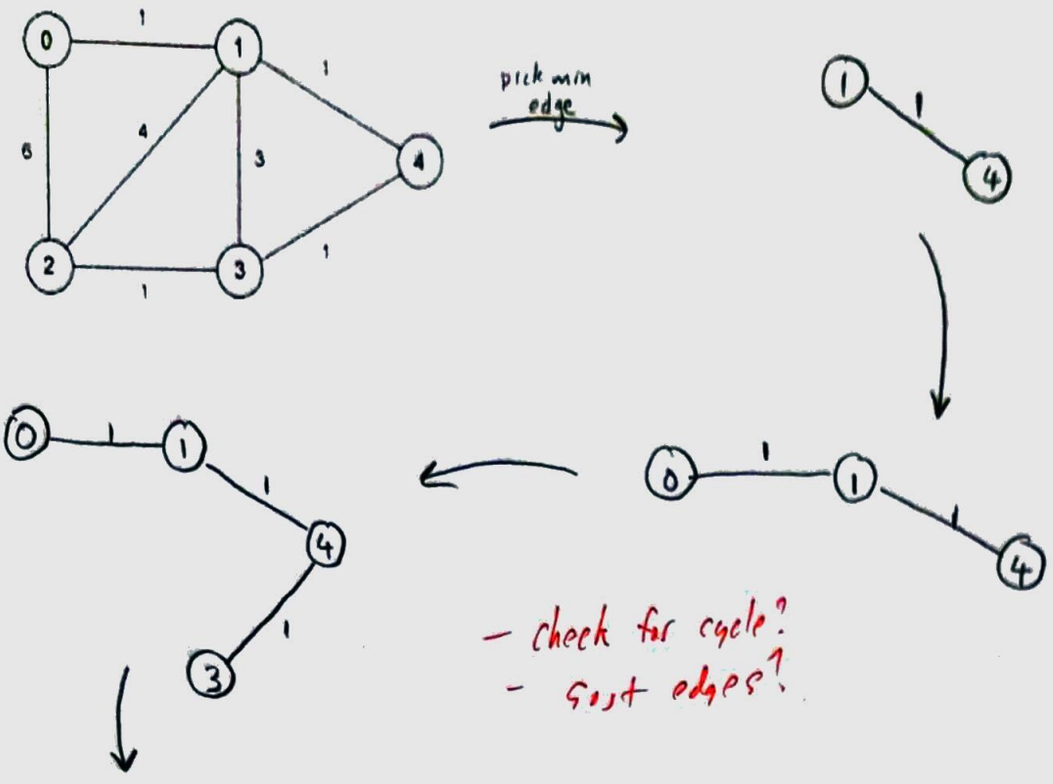
all have in&out deg of 1

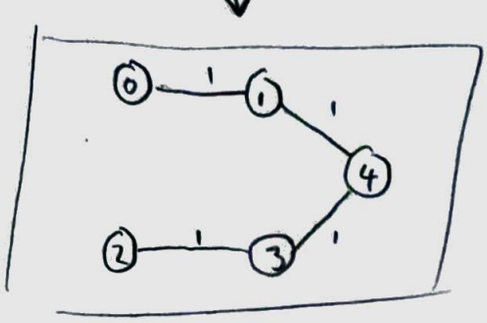
23

Name(last, first): Petcman, Ryan

**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)



B) Claim: Kruskal's generates an MST, even if there are negative edges

Proof (Induction on # of picked edges): Base: $n=1$, since Kruskals sorts the edge weights and picks the minimum edge the $n=1$ edge must be of minimum weight even if negatives allowed, since sorting allows for all real #'s to be ordered

Inductive Step:
Assuming we have picked the minimum edges without cycle up until the $i_{th}$ edge, show the $i_{th}+1$ edge will be minimum from the remaining.

Two Cases
(Trivial) 1. we pick the minimum edge of the remaining by nature of the algo. that does not cause a cycle ✓

2. we pick the next edge because the minimum remaining edge caused a cycle
In this case the vertices connected by the minimum edge were already connected by a more negative edge in an iteration before the $(i+1)^{th}$

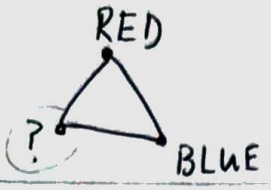
end on back

Name(last, first): Peterman, Ryan

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, ie.




B) Two Coloring Algo (altered BFS):

maintain a queue of nodes to visit

while all nodes not colored ← to handle multiple connected components
    push an arbitrary uncolored node on queue
    give it arbitrary color red or blue
    while queue is not empty
        take top node off queue
        for all adjacent nodes to top node
            if node is colored
                check if color is opposite of color of top node
                if not then there is no two coloring
            else if node not colored
                color it opposite color of top node
                push it onto the queue

if all nodes colored when algo terminates, graph is two colorable

Proof of Correctness (contradiction):

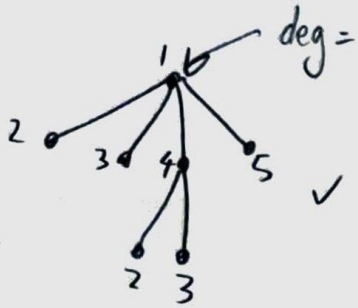
Assume that the algorithm does not produce a valid two coloring of a graph We know that by BFS we will visit/color every node exactly once. Also, by the algorithm we know we will visit each edge twice (once from each vertex). Since we will not color/visit a node twice the second time we use an edge we must check the coloring with the current coloring. Thus we cannot produce an invalid two coloring with this algorithm since we check every node atleast once.

Note: if no edges to vertex then its coloring is trivial

C. $O(V)$ since we iterate over all uncolored nodes
$O(E)$ since we iterate over all adjacent edges to a node
Thus, $O(V+E)$ ✓


D. In order to color a tree, we need one more than the degree of the highest degree node in the tree. This node has the most adjacent connections. If we satisfy its coloring then there will be more than enough colorings for the rest of the tree.

ie.




deg = 4, 5 colors needed ✗ ✓

-5

Name(last, first): Peterman, Ryan

22/25

**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.

maintain a current (tmp) sum, max (saved) sum #'s,

17 maintain i & j (tmp iterators)
maintain beg & end (saved iterators),
all initialized at 0

1. for all numbers in the list from the beginning
2. add the current # to current sum
3. if current sum > max sum
   max sum = current sum
   beg = i
   end = j
4. if current sum < 0
   current sum = 0
   i = j
5. j = j+1

what if 0 < currentsum < maxsum?

ex.

| | | | beg | | | | end | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
| | -2 | -2 | 5 | 7 | -3 | 4 | -4 | |
| i | 0 | 1 | 2 | 2 | 2 | 2 | 2 | |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
| curr | 0 | 0 | 5 | 12 | 9 | 13 | 9 | |
| max | 0 | 0 | 5 | 12 | 12 | 13 | 13 | |
| beg | 0 | 0 | 2 | 2 | 3 | 2 | 2 | |
| end | 0 | 0 | 2 | 3 | 3 | 5 | 5 | |

After, beg and end hold the indexes of i & j that correspond to the largest sum.

Proof on back →

We know beg & end corresponde to the indexes of the range of #'s resulting in max sum being set (Step 3)

(Claim) Thus, we must show that max sum holds the largest sum of consecutive integers in the list

6 Proof:

In the $i^{th}$ iteration curr_sum can be $\geq 0$ or $< 0$

- if currsum is $\geq 0$ we keep its value since the numbers in the sum are contributing a net positive value to the next curr sum
- if curr sum is $< 0$ we know it will only take away from the next curr sum and thus we set it back to 0

Thus, curr_sum greedily seeks to become more and more positive and max sum retains curr_sum's largest value