UCLA
Computer Science Department

CS180– Midterm
Algorithms & Complexity

10/30/2018

Name: [redacted]
UID: [redacted]

This exam contains 7 pages (including this cover page) and 6 questions.

- Writing has to be legible.
- Express algorithms in bullet form, step by step.

Distribution of Marks

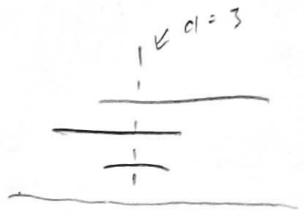
Question	Points	Score
1	20	18
2	20	17
3	20	14
4	10	10
5	20	5
6	10	6
Total:	100	70

Name(last, first): [redacted] Midterm 18 10/30/2018

1. (20 points) Consider a set of intervals $I_1, I_2, \cdots, I_n$:
   (a) Design a linear time algorithm (assume that intervals are sorted in any manner you wish) that assigns the intervals to the minimum number of processors.
   (b) Prove the correctness of your algorithm.

(a) → first, let's do a plane sweep, assuming intervals are sorted by start time.
→ we keep a counter, every time we encounter a start time we add 1, every time we encounter an end time we subtract one.
→ we save the maximum value that this counter achieves, let's call it d.
→ now, let's say we have processors $1, 2 \cdots d$,
→ taking each interval in order of start time,
→ assign to any open processor.

then we will have a schedule that maps to the minimum # of processors.



(b) proof:
→ suppose we could use less processors $d' < d$.
→ but, by our algorithm, we see that d is obtained by the number of starting times - end times we passed before a given time.
→ so, at some time, t, there will be d simultaneous tasks, which cannot be allocated to d' processors.

→ now we show that our algorithm provides a valid mapping, that is, there is always one or more open processors when we schedule a task.
→ suppose all d processors are full, and another task comes in to be scheduled.
→ then, we have $d+1$ simultaneous tasks.
→ so, this is a contradiction, as d would have been larger in our initial calculation.

17/20

2. (20 points) (a) Design an efficient algorithm that outputs the vertices of a DAG (Directed Acylic Graph), such that if there is an edge (x, y) then x is output before y.

(b) Analyze the run time of your algorithm.

(a) keep an in-degree V_i and out degree V_o for each vertex

for each edge (A, B) in G:

add 1 to A_o (out degree of A) ✓

add 1 to B_i (in degree of B) ✓

Now we have the indegree/out degree of each vertex.

while we haven't printed all vertices:

print a vertex V with in degree 0 ✓

for each of V's outgoing edges (V, V') 10/10

subtract 1 from in degree of V' ✓

$O(O+V)$

(b) it takes ~~$O(E)$~~ to calculate the indegree & out degree of each vertex.

it takes $O(V)$ time to find the 1st vertex with in degree 0. ✓

it takes $O(E)$ time to update all the degrees and find all ✓ vertices w/ in degree 0 after the first.

i.e we attribute the cost of updating indegrees and check if vertex has indegree 0 to the edges.) if we have an edge $e = (A, B)$, we just subtract 1 from B_i and check if $B_i = 0$, for each edge $e \in E$.

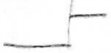
so, $O(E+V)$ time.

7/10

3. (20 points) An undirected graph is said to have property X if you can start from a vertex, traverse all edges of the graph exactly once, without removing your pen from the paper.
   (a) Classify the graphs that have property X?
   (b) Design an efficient algorithm for generating a traversal of a graph that has property X.

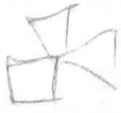
(a) a graph G has property X, if it has 0 or 2 vertices of odd degree. ✓

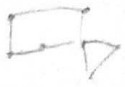
(b) let's run a DFS, starting at a vertex w/ odd degree, if it exists, otherwise, start at an arbitrary vertex.
during each step of DFS, we now pick an edge to explore, not a vertex.
So, we can repeat vertices but not edges.

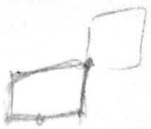


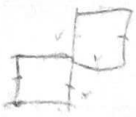
Keep a stack S, and an output list O
Start at any vertex V (current vertex = V)
push all edges of V to S
while S not empty:
  pop an edge (A,B) from S, append (A,B) to O, mark (A,B) as explored
  Set current vertex to B
  push all unexplored edges of current vertex to S, if they are not in S already
  if we didn't push any, remove last edge added to O and mark it as unexplored

-6



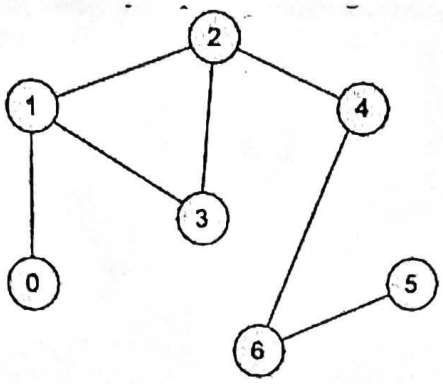




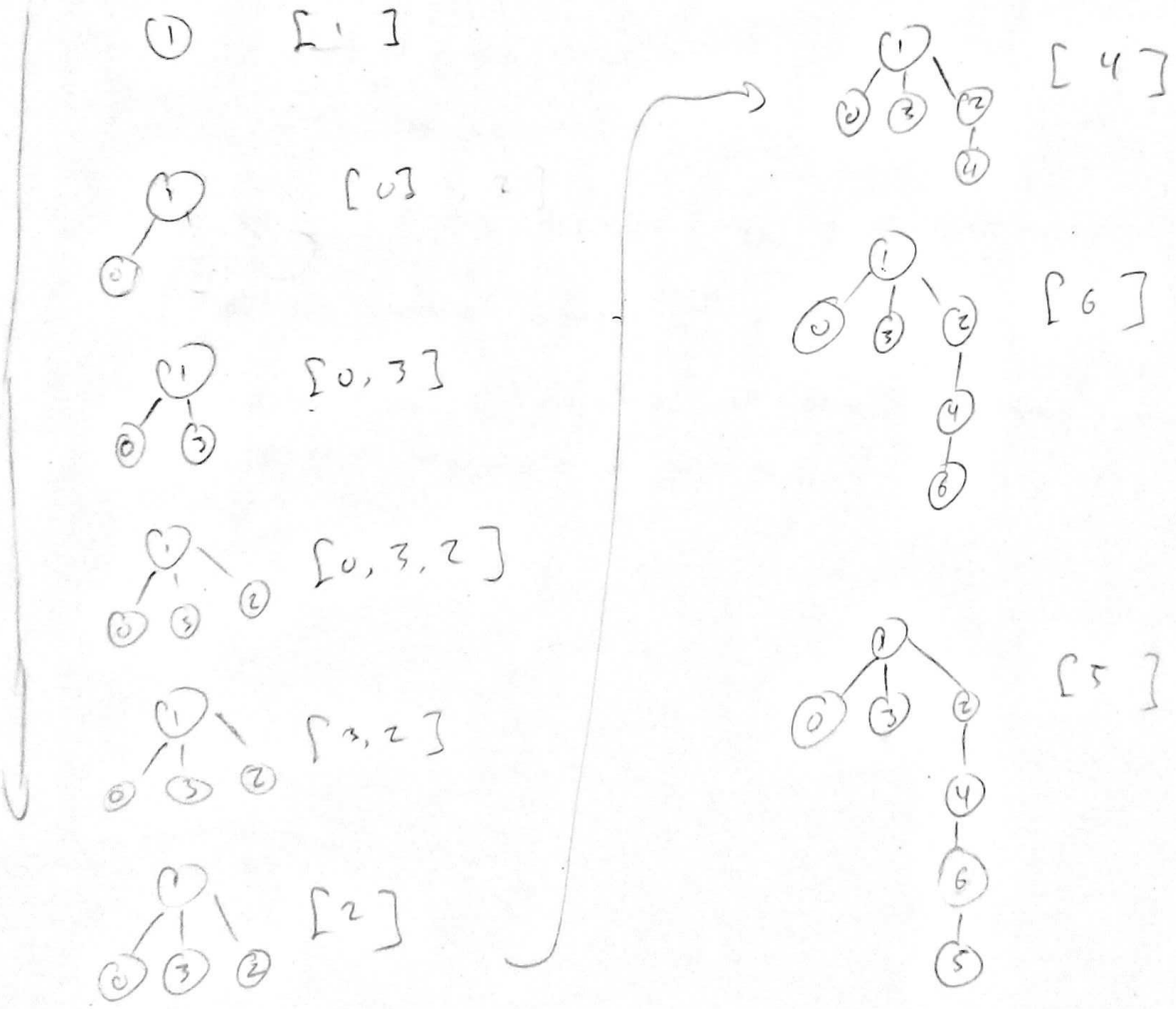


4. (10 points) Consider an unweighted graph G shown below:

   (a) Starting from vertex 1, show every step of BFS along with the corresponding FIFO next to it.



Start



end

5. (20 points) Consider an unsorted list of integers. You can find the minimum number in the list with $n-1$ comparisons. Similarly, you can find the maximum with $n-1$ comparisons. So you can find both the minimum and the maximum with about $2n-3$ comparisons. Design an algorithm that finds both the minimum and the maximum using about $\frac{3n}{2}$ comparisons.

max = list[0]
min = list[0]
for each integer i in list[1] ... list[n]
  if i > max: set max = i (note: if i > max, we don't check min)
  else if i < min: set min = i

$2n$

lets consider some cases:

① list is strictly increasing: n comparisons

② list is strictly decreasing: $2n$ comparisons

so, in average case, $\frac{n+2n}{2} = \frac{3n}{2}$ comparisons

intuition: about $\frac{n}{2}$ times we will have $i >$ max so, we don't have to compare it $<$ min.

6. (10 points) Give an algorithm to color a graph with 2 colors (assuming it is 2-colorable). A proof of correctness is not necessary.

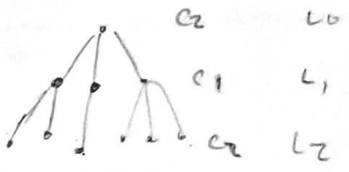
assuming the Graph is 2 colorable:

run BFS starting from any vertex, generating a tree T.

Now, color odd levels of the tree with color 1,

color even levels in the tree with color 2.

Then the graph is 2 colored.



-4 Time complexity?