UCLA
Computer Science Department
CS180– Midterm
Algorithms & Complexity

10/30/2018

Name:
UID:

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.

Zhang.

Friday 12-2

Distribution of Marks

Question	Points	Score
1	20	20
2	20	17
3	20	14
4	10	10
5	20	20
6	10	10
Total:	100	91

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.

- Let's say that the start time of each interval I_i is s_i and the finishing time is f_i
- we can sort s_i and f_i in a list.
- for every Interval I_i:
  - take I_i with the earliest s_i in the list and assign it to a free processor. (when f_i is over a processor will be freed up).
  - Remove I_i from consideration.

| L | L | L | R. |

proof:
- Lets say that A $= \{a_1, \ldots, a_n\}$ is the result of our greedy algorithm and
- O $= \{o_1, \ldots, o_n\}$ is the result of the optimal algorithm
- let's assume that the first k tasks assigned were the same.
- O chooses to assign a task that starts at $s(o_{k+1})$ while A choose to assign a task that starts at $s(a_{k+1})$.
- We know that if $a_{k+1} \neq o_{k+1}$ $s(a_{k+1}) < s(o_{k+1})$ due to the way tasks are assigned.
- Therefore, if we swap a_{k+1} we can still complete the rest of the tasks since $f(a_{k+1}) < f(o_{k+1})$, we even free up the processor earlier.
- Inductively, we can keep swapping a_i with o_i to get an algorithm no worse than the greedy algorithm.
- Hence, the greedy algorithm is the optimal algorithm.

Name(last, first): [redacted] Midterm $\frac{17}{20}$ 10/30/2018

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)

- This is a topological sort.
- Count the # of incoming edges on to each vertex, let's call this the degree.
- Choose an arbitrary node with 0 incoming edges.
- Add node to sorted list S, delete n from the graph and decrement the degree of neighboring vertices by 1.

→ repeat until all vertices are deleted. $\frac{10}{10}$

- output S.

proof by induction:

base case: this algorithm holds true for a DAG of 1 or 2.

- 1 → it just outputs the node.
- 2 → it outputs the source and then the node it is connected to.

Inductive step: Let's assume this algorithm holds for n-noded DAG.

for n+1: On the n^{th} nodes turn, it will delete all edges that it is connected to. Making the degree of the $n+1^{th}$ node become 0 as it is in the next level. n+1 is then added to the already topologically sorted list.

- S remains topologically sorted as n → n+1.

b)

- It takes O(E) time to count all incoming edges. → when?
- It takes O(V) time to go to, add to S and delete each node →
- It takes O(E) to delete all edges. ✓

∴ O(V+E) ✓

$\frac{7}{10}$

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) All graphs that have property X are called Eulerian graphs.
The unifying property behind these graphs is that the have at most 2 vertices w/ odd degree. ✓

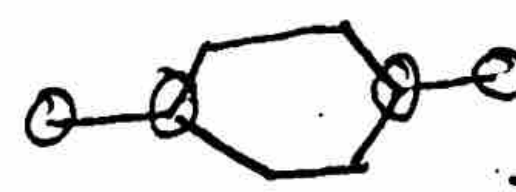
cases

- 0 odd vertices: A graph with 0 odd vertices is a cycle and by definition it has a path by which it can return to itself.

- 1 odd vertex: Impossible. We know $\sum$ degrees = 2·#edges. since $\sum$ degrees must always be even we can never have just 1 odd vertex as that would result in an odd summation.

- 2 vertices: There is a way to enter and a way to exit the even vertices.

- odd:



does not work.
- There is no way to reenter after 2 odd ~~vertices~~. enter exit no reenter.

b)
- pick an arbitrary node
- move to an arbitrary neighbor
- delete traveled

repeat until all nodes have been reach / no edges left to travel.

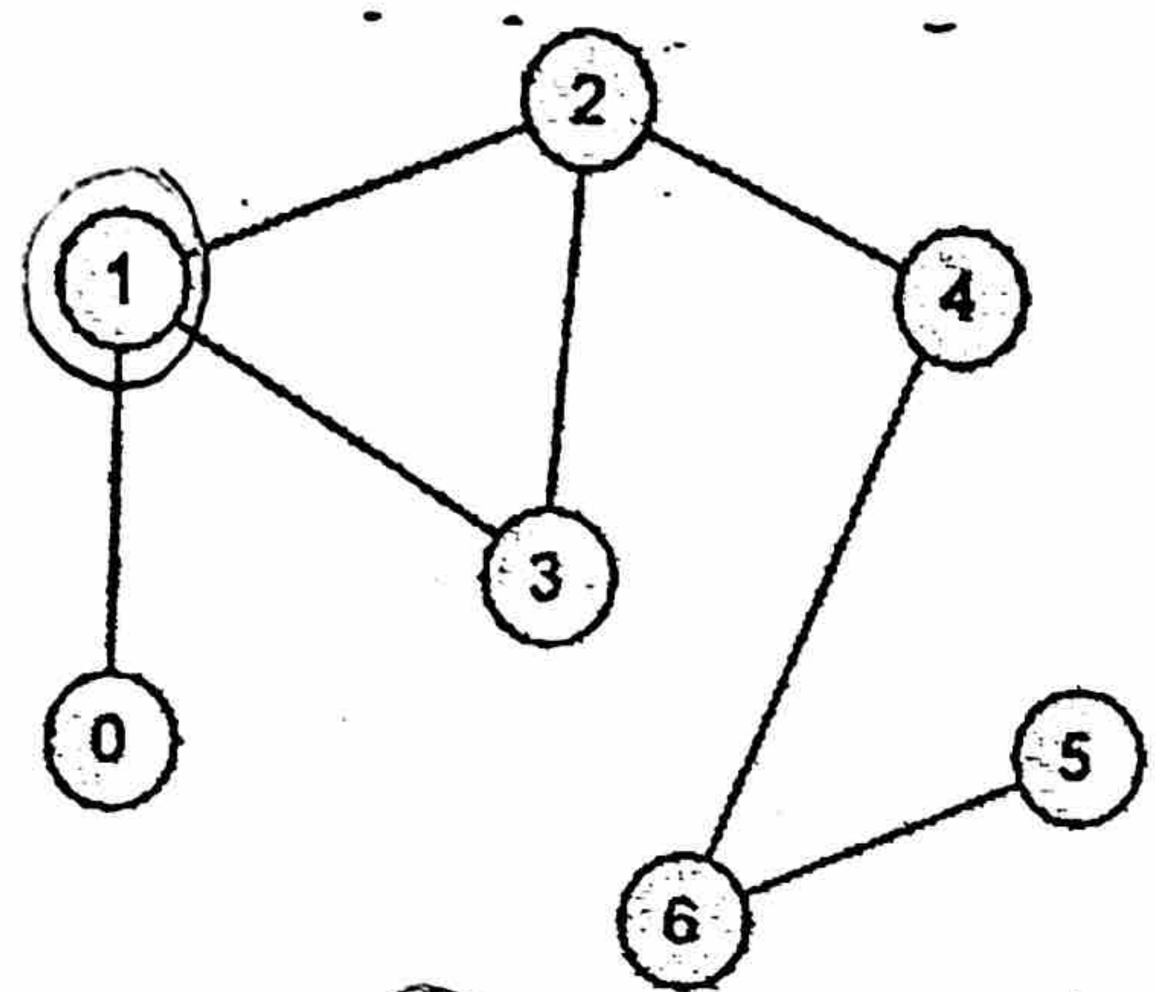
-6

∴ O(E) travelling and deleting edges.

proof: Suppose V is a node that alg did not traverse through.
- this must mean there is no edge e in the graph that connects the rest of the graph to v as edges are only deleted if nodes are traversed.
- G is not a Eulerian graph ↯.

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.



step 1 add 1 to queue — 1

step 2 pop 1 - mark as visited

step 3 add unvisited neighbors of 1 to queue — 2, 3, 0

step 4 pop 0 - mark as visited — 2, 3

step 5 add unvisited neighbors of 0 — 2, 3

Step 6 pop 3 - mark visited — 2

step 7 add unvisted neighbors of 3 — 2

step 8 pop 2 - ~~mark~~ mark visited

step 9 add unvisited neighbors of 2 — 4

step 10 pop 4 - mark visited

step 11 add unvisited neighbors of 4 — 6

step 12 pop 6 - visited

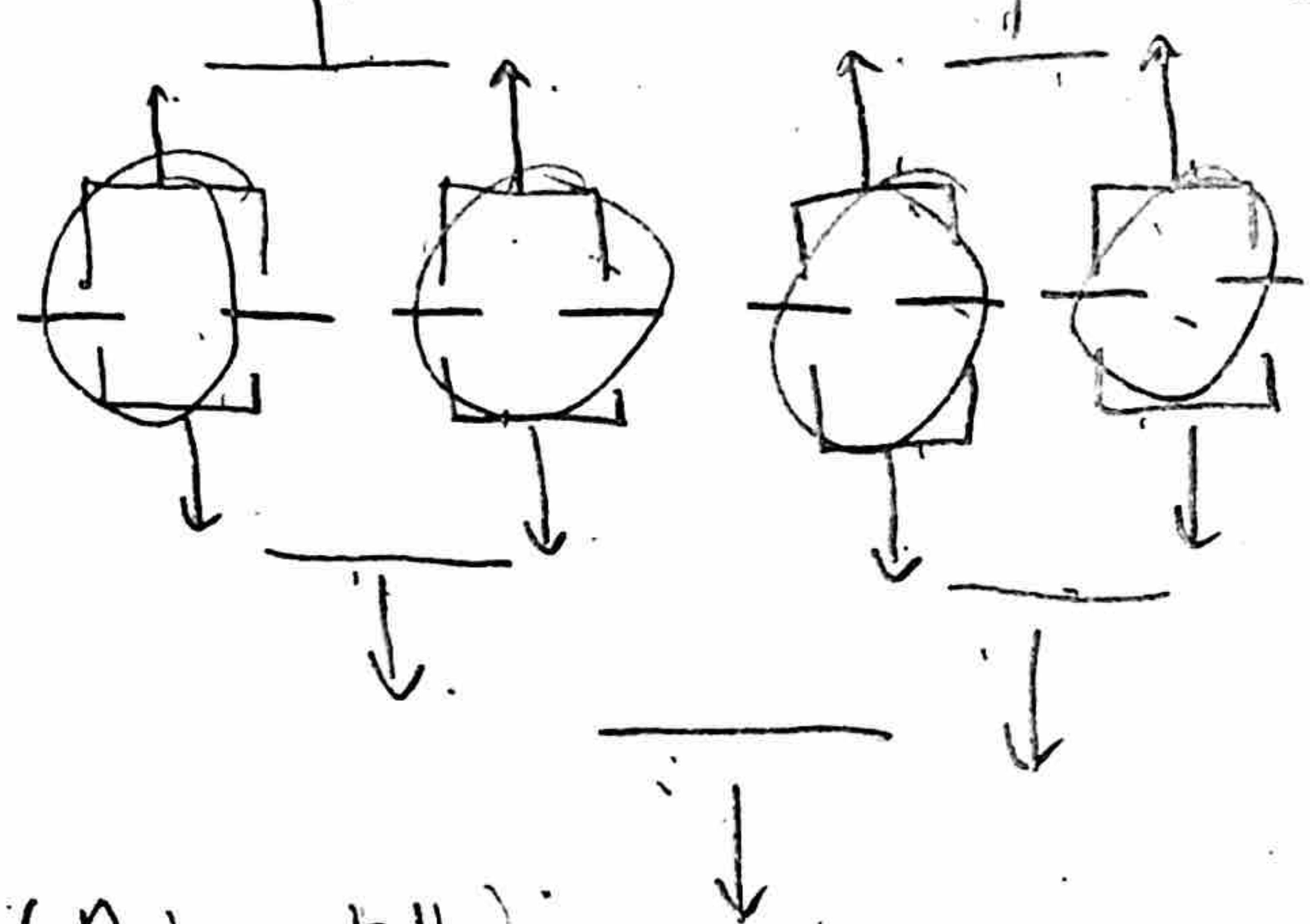
13 add unvisited neighbors of 6 — 5

14 pop 5 - visited

15 add unvisited neighbors

16 queue empty! we are done.

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.



$\frac{3 \cdot 8}{2}$ $1+2+$

$1+2+4+8+2^n = \frac{n}{11}$

$\frac{r^{n+1}-1}{(r-1)}$

$\frac{n}{2} + 2(\underbrace{\frac{n}{4} + \dots + 1}_{n/2}) = \frac{3n}{2}$.

Divide and conquer.

- compare first term w/ second. third w/ forth ... $n-1$ w/ n.
- <u>now</u> compare the larger terms of 1,2 and 3,4 ... $n-3, n-1, n$
- Keep going recursively until only 1 term is left. comparing for greater term.
- <u>now</u> compare the smaller terms of 1,2 and 3,4 ... $n-3, n-$ and $n-$
- keep going recursively until only 1 term is left. comparing for lesser term.
- The two resulting #s are the greatest and smallest numbers

<u>proof</u>: suppose a_1 = alg greatest.
a_2 = alg smallest.

- a_1 is not the greatest in the list.
- this is a contradiction as that must mean the real greatest failed a comparison test $\to$ recursive rounds.
- a_2 is not the minimum in the list.
- This is a contradiction because it means that the real minimum failed a comparison test

<u>comparisons required</u>:

$n/2 \to$ first round.

$2(n/4 + n/8 \dots + 1) \sim n$

$\sim 3n/2$.

6

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.

- We can pick an arbitrary node n and run BFS from it.
- Every odd level, we color the nodes color 1 and every even level we color the nodes color 2.
- We then run a check on all edges and nodes to make sure that no 2 edges are adjacent to the same color node.
  - if we find 2 same colored adjacent nodes:
    - the graph is not 2 colorable.
  - if we do not:
    - we have successfully 2 colored the graph.

This algorithm runs in O(V+E)

- O(V+E) for BFS
- O(V) to color nodes.
- O(V+E) to check.