Sujash Saxena

15

Name(last, first): Midterm 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.

- Sort intervals in acending order of start time.
- Let {1, 2 ... d} be the available set of procs. (Minimum = d i.e. no. of clashing intervals)

do path? how to find it?

- For each 'j' from 1 to n:
  - For each interval i (in sorted list) < j that ends after Ij:
    - Mark off a processor in available set. (ie. unavailable)
  - If there are available processors in set:
    Assign any one to I_j.
    Else leave I_j unassigned (for now).

order is not analyzed

Proof:

- Clashing intervals are not assigned same processor. If K tasks clash w arbitrarily task j, K processors are removed from sample set. We pick one from d-k processors.
- Each interval is assigned a processor: At any given instance, at most d intervals clash, so, an arbitrary task clashes w

Sujash

20/20

Name(last, first): Midterm 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.

① Construct a map of vertices to their in-degree i.e. no. of incoming edges, by traversing the DAG.

② Find a vertex w/ in-degree = 0. Output this 'src' in solution.

③ Remove 'src' from map of in-degrees.
for each edge (src, adj):
Decrement in-degree of adj by 1.

④ Repeat from ② until all vertices have been outputted.

10/10

Analysis:

① O(E) traversal

② O(V) to find first source

③ O(E) steps to change ALL E in-degrees

④ O(V-1) repetitions to output all $v \in V$.

∴ Run-time ⟹ $O(V+E)$

10/10

Sayash 17

Name(last, first): Midterm 10/30/2018

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) Graphs that have an even no. of odd-degree vertices show property X. Further, the no. of such odd-degree vertices must be 0 or 2. ↓ cycle ↓ start-end points.

-3

b.) - Pick an arbitrary src and traverse the graph counting the degree of each vertex. (w/o double counting for undirected edges)

- If an vertex not visited ⇒ Disconnected ∴ No Prop X

- Find vertices of odd degree
  - If 2 vertices found, say S and E:
    - Perform modified DFS from S such that edges cannot be retraversed. Output path will traverse all edges once and end at E.

keep hashset of (v,w) and (w,v) visited edges.

  - If no vertices found:
    - Pick arbitrary src
    - DFS as above, output path from src to src (ie. cycle w/ all edges)
  - Else no such traversal exists.

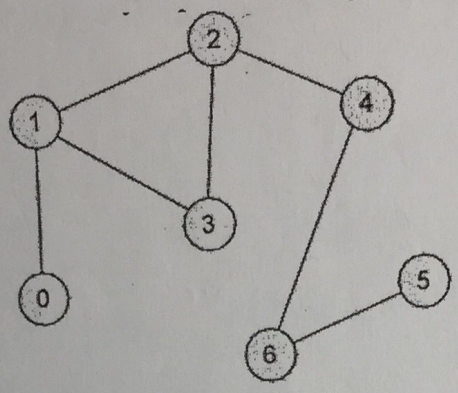
O(E)

Sujash 10

Name(last, first): Midterm 10/30/2018

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.



⓪ $Q = \{\}$

① $Q = \{1\}$

visit 1 (and pop)

② $Q = \{2, 3, 0\}$

visit 2.

③ $Q = \{3, 0, 4\}$

visit 3

④ $Q = \{0, 4\}$ visit 0

⑤ $Q = \{4\}$ visit 4

⑥ $Q = \{6\}$ visit 6

⑦ $Q = \{5\}$ visit 5

⑧ $Q = \{\} \Rightarrow$ STOP

∴ Traversal :

1
2
3
0
4
6
5

Sujash

Name(last, first): Midterm 10/30/2018

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.

$= n + \frac{1}{2}n$ 1.5 n vs. 2n - 3

- Keep 2 lists, one for tracking max = BIG, one for tracking remaining = SMALL.
- Set Max = 1st element, BIG = {1st element}.
- For each remaining element E in list: ← n-1 steps
  - if (E > max):
    - Append E to BIG, update max = E.
  - else
    - Append E to SMALL

- Max element is found.
- If SMALL is empty, Min element is first elmnt of BIG.

Else:

Perform a linear search in SMALL to find the min.

PTO

Sujash

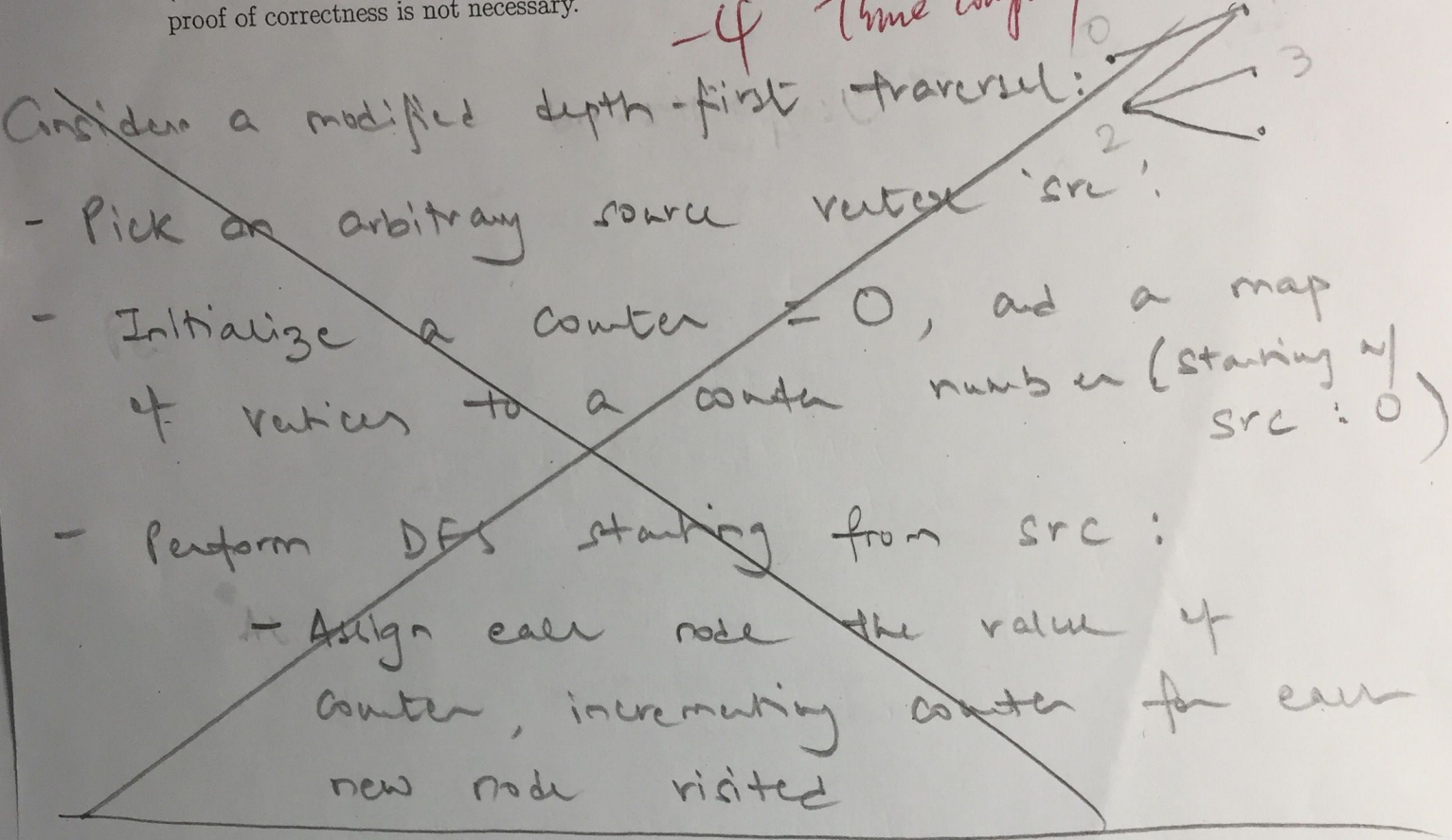
6

10/30/2018

Name(last, first): Midterm

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.

-4 Time complexity



Consider a modified depth-first traversal:

- Pick an arbitrary source vertex 'src'.
- Initialize a counter = 0, and a map of vertices to a counter number (starting w/ src : 0)
- Perform DFS starting from src:
  - Assign each node the value of counter, incrementing counter for each new node visited
- Pick arbitrary vertex 'v', assign color A to it.
- for each vertex 'adj' adjacent to it, that does not already have a color:
  - Assign opposite color to adj i.e B.
  - Repeat (via recursion / stack in DFS fashion) for v = adj.
- If all vertices are not colored: (unconnected)
  Repeat from top for arbitrary uncolored vertex until all are assigned a color.

Sujash

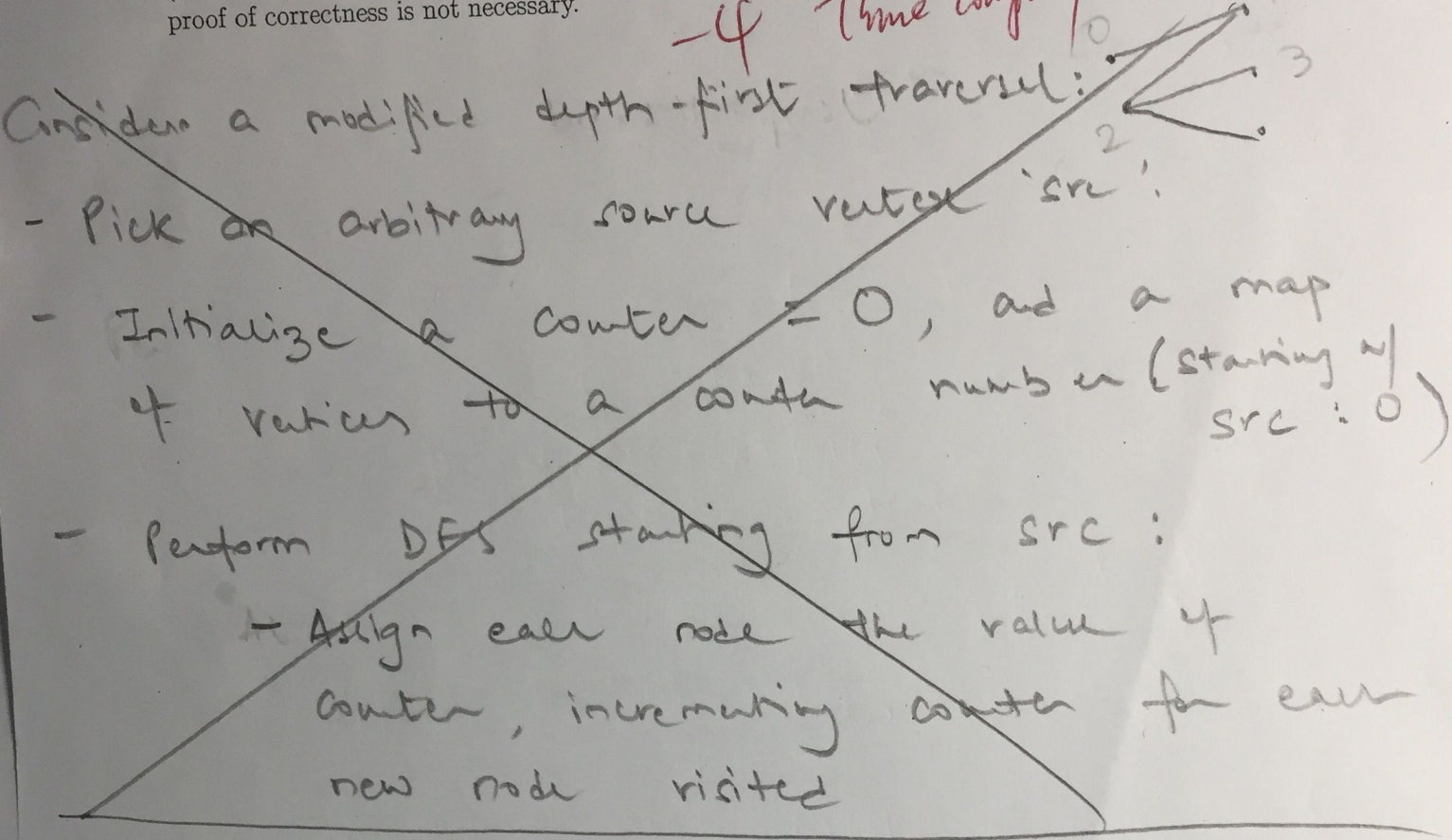
6

10/30/2018

Name(last, first): Midterm

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.

-4 Time complexity



Consider a modified depth-first traversal:

- Pick an arbitrary source vertex 'src'.
- Initialize a counter = 0, and a map of vertices to a counter number (starting w/ src : 0)
- Perform DFS starting from src:
  - Assign each node the value of counter, incrementing counter for each new node visited
- Pick arbitrary vertex 'v', assign color A to it.
- for each vertex 'adj' adjacent to it, that does not already have a color:
  - Assign opposite color to adj i.e B.
  - Repeat (via recursion / stack in DFS fashion) for v = adj.
- If all vertices are not colored: (unconnected)
  Repeat from top for arbitrary uncolored vertex until all are assigned a color.