/

UCLA
Computer Science Department

CS180– Midterm
Algorithms & Complexity

10/30/2018

Name: ____________________ Friday 4pm
UID: ____________________ TA: Eanri ID

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	20
2	20	20
3	20	10
4	10	10
5	20	20
6	10	6 +4
Total:	100	86

90

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) Sort the intervals by start time. Let there be d = max crossing level, # of processors
1 - Iterate forward through the list of intervals
2 - assign this interval to an arbitrary processor that is available
One pass is O(n)

b) Proof: Say the max crossing level is d. Then at time t_d we need d processors to handle all those simultaneous intervals. d = minimum # of processors.
Our greedy algo only has d processors available, so we have to prove that step 2 always finds an available processor
Proof by contradiction: Assume step 2 does not find an available processor.
Let the interval start at t_k.
At t_k, we have already assigned all intervals starting before t_k, because we are in sorted order.
If there is no available processor at time t_k then there are already d intervals being processed.
At t_k we would add this interval, so d+1 simultaneous intervals would occur, which is a contradiction.

$\frac{20}{20}$

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) We use topological sort

1 - Initialize a list of the incoming degree of every vertex, i.e. # of edges ✓ that end at that vertex, with all counts set to 0 ✓

2 - Iterate over every edge, add 1 to the count of incoming degree for ✓ the destination node

3 - Iterate over the list of in-degrees add all sources (degree = 0) to ✓ a queue

4 - Initialize an empty output list of the vertices in order ✓

5 - while the queue is not empty: ✓

   6 - pop from the queue, WLOG name this vertex v ✓

   7 - examine all outgoing edges from v

   8 - decrement the degree of the destination by 1 ✓

   9 - if the destination degree is 0, add it to the queue

   10 - add v to the output list

11 - return the output list

$\frac{10}{10}$

b) Step 1 is ~~[illegible]~~ O(n), Step 2 is O(e), Step 3 is O(n), Step 4 is O(1)

The loop at step 5 is run n times, assuming the topological sort order exists, every node will be added exactly once. Within this loop, step 6, 8, 9, 10 are O(1). Step 7 will examine every edge exactly once so instead of O(n²) it is O(e).

Overall, this is O(m+n) ✓

$\frac{10}{10}$

10

Name(last, first): Mitra, Nihar   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) The graph has a Euclidean path. This means the graph is connected and has an even # of nodes with odd degree. $\longrightarrow \leq 2$

-4

b) A DFS will traverse every edge exactly once if it starts at the correct node. We assume that the graph has property X.

- Initialize a stack of nodes and a list of whether an edge has been visited
- Make a list of the degree of each vertex, initialized to all 0's
- Iterate over all the edges and update the degrees accordingly
- Iterate over the list of degrees to find the smallest odd degree
- Add that node to the stack, if no such node exists then pick an arbitrary node
- Create a stack of edges to represent our path taken
- Repeatedly pop the stack of nodes
  - pick an arbitrary unvisited edge,
  - add that edge to the stack of edges
  - mark the edge as visited
  - add the destination node to the stack of nodes
  - if this vertex has no unvisited edge, backtrack the path until we find an alternate route, i.e. choosing a different unvisited edge at a previous node
- Finish the algorithm when all edges are traversed (check this before backtracking)

-6

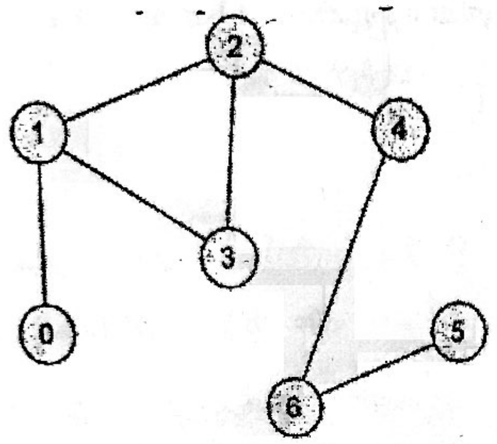
Justification: we must start at an odd vertex, picking the smallest odd allows us to start at a dead end to reduce backtracking. This is because odd vertices lead to cycles so if they exist we must start or end at one. The algo is optimal because with sufficient backtracking it will find the cycles and build a path that orders the edges correctly.

The efficiency of this algorithm is polynomial as in the worst case it will perform a DFS on every node and backtrack to another node. O(mn)

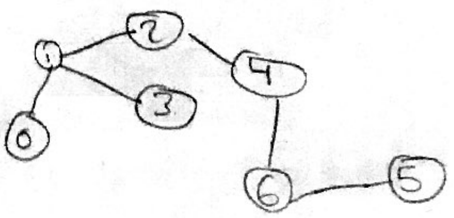
10

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 at vertex 1, add 1 to the queue. queue: [1]
Pop the queue, add all unvisited neighbors: queue [0, 2, 3]
Pop the queue, add all unvisited neighbors: queue [2, 3]
Pop the queue, add all unvisited neighbors: queue [3, 4]
again, queue: [4]
again, queue: [6]
again, queue: [5]
again, queue: [ ]
queue is empty, terminate.



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.

We can modify the concept of the famous /celebrity problem and reduce the problem size:

- Start with the initial unsorted list, and 2 empty lists: one for candidates to be minimum and one to be candidates for maximum
- Iterate over pairs of consecutive integers:
    - for each pair, compare the integers
    - send the smaller integer to the list of minimum candidates
    - send the larger integer to the list of maximum candidates
- if the list has an odd number leave the odd one out in the list
- then for the list of minimum candidates
    - store a temporary minimum = the first candidate
    - iterate over the list and compare each integer to the temp. min.
    - if smaller, update the temp. min
- likewise for the list of maximum candidates
    - store a temp. max = the first candidate
    - compare every other candidate to the temp max, updating it as needed
- if there was an odd one out in the original list, compare it to the temp min and temp max and update accordingly
- the temp min and max are returned.

Runtime analysis: the first list has $\frac{n}{2}$ comparisons (for the $\frac{n}{2}$ pairs). Then the two subsequent lists are $\frac{n}{2}$ long, and each add $\frac{n}{2}$ comparisons. For a total $\sim \frac{3n}{2}$

6+4

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.

Modified BFS:

WLOG let the colors used be a and b

```
- Start at an arbitrary node, color it as a, add it to a queue
- While: the queue is not empty
    - pop the queue, say vertex WLOG is named v
    - examine each neighbor of v:
        - if the neighbor has no color assigned,
            - add the neighbor to the queue
            - if v is color a, color the neighbor with b
              otherwise, color the neighbor with a
```

~~-4~~ Time complexity?

Grading Error, no points deducted