UCLA
Computer Science Department

CS180– Midterm
Algorithms & Complexity

10/30/2018

Name: ZHUOYU JI Friday 2-4
UID: 20499291

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	14
4	10	10
5	20	10
6	10	6
Total:	100	80

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 we calculate the density of all events, which is the maximum number of events that are overlapping at a certain time t.

- We calculate the event density by initializing a counter C, with initial value 0.
- Each event is represented by two points, start and end, on the time line.
- Starting from the leftmost start point (first start point of all intervals), we scan the timeline from left to right.
- Whenever we encounter a start point we increment C; whenever we encounter an end point we decrement C.
- As we are running this process we always keep track of the maximum value of C it has ever reached, denoted as C_{max}
- After we've processed all intervals, C_{max} should be the event density as well as the minimum number of processors needed so that all intervals can be assigned.
- We then arbitrarily pick an interval and assign it to an arbitrary processor which is free. We're guaranteed that all intervals can be assigned successfully in this way.

✓

(b) . With C_{max} number of processors, we can always guarantee that at least one processor is free to take an arbitrary interval at that time.

. Because if it is not true, it means that all C_{max} processors are occupied at the time when a new interval needs to be assigned, implying that there are already C_{max} number of events processed concurrently. Adding one extra event we are about to process now, we have $C_{max}+1$ number of intervals concurrently, which is impossibly as it contradicts the definition of C_{max}, which is event density. Thus it is always true.

✓

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) • Initialize the graph by assign a number I that denotes the number of incoming edges for each node. Keep another list L which includes all nodes which have I = 0 ✓ ✓

• While L is not empty ✓
  • we arbitrarily pick a node in L, then it must have I = 0 ✓
  • we output the node in the ordering ✓
  • we delete the node and all its outward pointing edges before we decrement the I count of all nodes that it points to by 1 ✓ ✓
  • If after the decrement the I count of some nodes become 0, we add them to L

$\frac{10}{10}$

• As we're guaranteed that a DAG always has a topological ordering, the previous algorithm would always work.

(b) • The initialization takes O(m+n) time, as we go through each node once, and each edge pointing to a node is counted once ✓ ✓

• While we run the while loop, each time we process a node and several edges that is points outward. However, each edge would be processed once, corresponding to one decrement of I count of the pointed node. Thus the while loop takes O(m+n) time

• The algorithm takes O(m+n) time

$\frac{10}{10}$

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 0 or 2 odd degrees node (degree means number of edges that a node connects to) ✓

This is because an odd degree point has to be a start or an end point

Proof: If it is an intermediate point, then each time we arrive must correspond a time we leave, thus the total degree should be even, which is contradictory

(b) · If the graph has 0 odd degree node

- We start from an arbitrary node, then choose an arbitrary edge connecting it and traverse along that edge before we marker that edge as explored.
- Each time when we reach one node, we pick an unexplored edge that it connects to and continue traversal along that edge.
- When we reach the starting node the traversal is completed

-6 · If the graph has 2 odd degree nodes

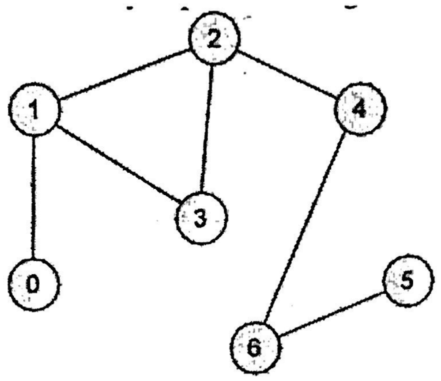
- We start from an arbitrary odd-degree node
- We repeat the same process as in the situation when the graph has 0 odd degree node
- When we reach the other odd degree node the traversal is completed.

· The algorithm always works. The reason why it works is:
Besides the starting node and end node, the intermediate nodes all have even degree. For all intermediate nodes, we always arrive the node first by an edge, and then immediately leave by another edge. Thus each pair of arrival and departure consumes 2 edges. As each intermediate nodes has even degree, each time when we arrive there is guaranteed to have one edge for us to leave, until we reach the end point.

10

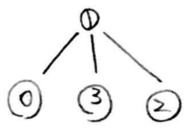
4. (10 points) Consider an unweighted graph G shown below:

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



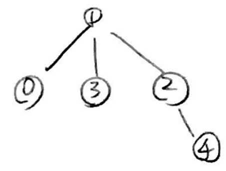
(a) Layer 0 : ① FIFO: 1

Layer 1 :



FIFO: 2 3 0 ~~1~~

Layer 2 :

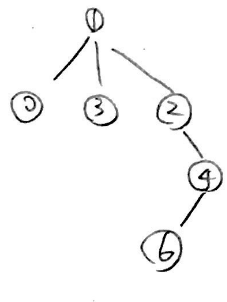


FIFO: 2 3 ~~0~~

2 ~~3~~

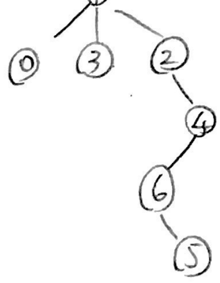
4 ~~2~~

Layer 3 :



FIFO: 6 ~~4~~

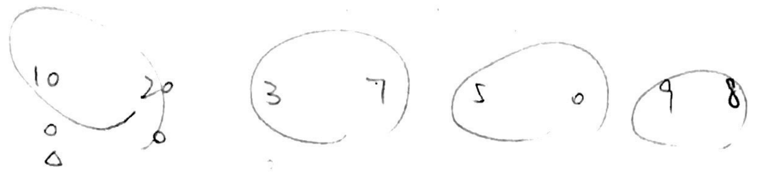
Layer 4 :



FIFO: 5 ~~6~~

lastly : FIFO: ~~5~~ and the queue is empty, we finish

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.



for every pair a_{2k} and a_{2k+1}, pick the bigger one and leave sma

then ?

10

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.

Assume the colors are A and B. We start by picking an arbitrary node and color it A, without loss of generality

- While not all nodes are colored
    - We run a BFS search starting from the first node picked (denote the Layer containing this node only as Lo)
      We color all odd Layers with B and even layer with A

-4 Time complexity