CS 180: Introduction to Algorithms and Complexity
Midterm Exam

May 6, 2020

Name	James Youn
UID	505399332
Section	Lec 1 Dis 1I

1	2	3	4	5	Total

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

★ **Your Exams need to be uploaded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.**

- There are 5 problems.
- Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and why it works.
- Your answers are supposed to be in a simple and understandable manner. Sloppy answers are expected to receive fewer points.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.

1. For each of the following problems answer True or False and briefly justify you answer.

   (a) (5pt) For a connected and undirected graph G, if removing edge e disconnects the graph, then e is a tree edge in DFS of G.

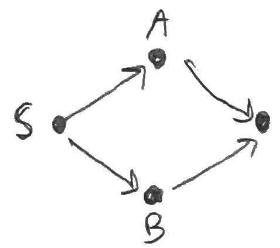
   (b) (5pt) For a DAG G, if there is only one node with no incoming edge, then there exists only one topological ordering.

   (c) (5pt) For the stable matching problem, if there is a man m_1 and woman w_1 such that w_1 has the lowest ranking in m_1's preference list and m_1 has the lowest ranking in w_1's preference list, then any stable matching will not contain the pair (m_1, w_1).

   (d) (5pt) If we run DFS on a DAG and node u is the first leaf node in the DFS tree, then u has no outgoing edge.

1a) Suppose, by contradiction, that e is not an edge of the DFS tree. Then, removing it does not disconnect G because the nodes are still connected by the DFS tree. ∴ e must be a tree edge. **True**

1b)



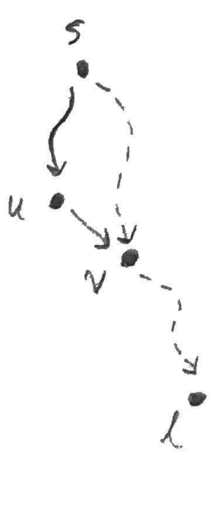
The graph is a DAG and s is only node with no incoming edge, but there is more than one ordering because either A or B can come before the other, due to symmetry. **False**

1c)

People	Preference (high to low)
m_1	w_2, w_1
m_2	w_2, w_1
w_1	m_2, m_1
w_2	m_2, m_1

In this case m_1 & w_1 end up together. If m_1 proposes to w_2 before m_2, then once m_2 proposes to w_2, w_2 will choose m_2. If m_2 proposes first, then w_2 will not change to m_1. **False**

1d) Suppose that DFS was run on node s and it eventually reached u. Suppose, by contradiction, that u has an outgoing edge to v.



v can be either visited already or not. If not, then u is obviously not the leaf node because u is not the end of the path that contains s & u. If v is visited, then there must be a path p that contains s & v, and the node that finishes p (node l) is the first leaf node. ∴ u must have no outgoing edges. **True**

2. (10pt) Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

log in base 2

- $f_1(n) = 3n^3$
- $f_2(n) = n(\log n)^{100}$
- $f_3(n) = 2^{n \log n}$
- $f_4(n) = 2^{\sqrt{n}}$
- $f_5(n) = 2^{0.8 \log n}$

$g_3(n) = \log(f_3(n)) = n \log n$

$g_4(n) = \sqrt{n}$

$g_5(n) = 0.8 \log n$

$g_1(n) = C + 3\log n$, C is a constant ($\log 3$)

$\log n = O(n^p), p > 0$. $(\log n)^{100} = O(n^{100p}) = O(n^r), r > 0 \therefore (\log n)^{100} = O(n^2) \therefore$

$f_2(n) = O(f_1(n))$

$\log n = O(\sqrt{n})$ & $\sqrt{n} = O(n) = O(n \log n) \therefore f_5(n) = O(f_4(n))$ & $f_4(n) = O(f_3(n))$

↑ The above reasoning only works because the function types are different.
If $g_3(n) = n$ & $g_4(n) = 2n$, then although $g_4(n) = O(g_3(n))$, $2^{g_4(n)} \neq O(2^{g_3(n)})$

$f_5(n) = n^{0.8} \therefore f_5(n) = O(f_2(n))$

$g_1(n) = O(g_4(n)) \therefore f_1(n) = O(f_4(n))$

f_5, f_2, f_1, f_4, f_3

3. (20pt) For a DAG with n nodes and m edges (and assume $m \geq n$), design an algorithm to test if there is a path that visits every node exactly once. The algorithm should run in $O(m)$ time.

Idea: Because graph is a DAG, there is a topological ordering. Topologically sort the graph. If for every nodes u & v such that u directly precedes v in the ordering, (u, v) ∈ E, then there exists such a path. If not, then there is no such path.

Proof: The first part of claim is obvious. However, if there is a u & v ∧ (u directly precedes v) such that (u,v) ∉ E, then there is no such path. By contradiction, if there were such a path, then u must precede v. But, since u can't go directly to v, it must first go to a node that is after v in the ordering. However, it is impossible that there is an edge from that node to v as this violates definition of topological order. ∴ no such path

Algo:

G is DAG

G' ← G (make copy of G)

- while there are unordered nodes:
  - Find v with no incoming edges and append to ordering
  - If u is preceding node in ordering & (u, v) ∉ E:
    - there is no such path
  - Delete v and its outgoing edges from G'
- There exists such a path

Topological ordering takes $O(n+m)$, but $m \geq n$, so $O(m)$

Assumed that index starts from 0 & up to $n-1$

4. (20pt) Given an array A of n distinct integers and assume they are sorted in increasing order. Design an algorithm to find whether there is an index i with $A[i] = i$. The algorithm should run in $O(\log n)$ time.

Idea: If $A[u] < u$, then $A[i] \neq i$ such that $i < u$. If $A[i] = i$, $i < u$, then minimum possible value for $A[u]$ is $A[i] + u - i = u$ ($A[u] \geq u$), which contradicts $A[u] < u$. By similar reasoning, if $A[u] > u$, then $A[i] \neq i$ for $i > u$. This allows us to restrict our search to either the portion less than or greater than u.

Algo: u got replaced with m.

l: denotes left most index of search portion

r: denotes right most index

```
while l ≤ r :
    m ← ⌊(r−l)/2⌋ + l
    if A[m] = m :
        there is index i such that A[i] = i
    if A[m] < m :
        l ← m+1
    if A[m] > m :
        r ← m−1
there is no such index
```

Because algo halves the search portion on every iteration, time complexity is $O(\log n)$

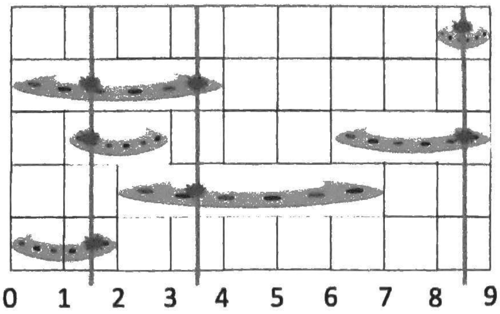


Figure 1: In this example, there are 6 flying saucers with $(L_1,R_1) = (0,2)$, $(L_2,R_2) = (2,7)$, $(L_3,R_3) = (1,3)$, $(L_4,R_4) = (6,9)$, $(L_5,R_5) = (0,4)$, $(L_6,R_6) = (8,9)$. We need at least 3 laser canons to destroy all of them, and 1.5, 3.5, 8.5 is a set of valid positions of these laser canons.

Assumption: (0,2) & (2,7) are noninclusive i.e. don't include 0, 2, or 7.
ε: Really small #

5. (30pt) There are several flying saucers on the sky to attack the Earth. For simplicity, we assume Earth surface is 1-D and the flying saucers are on the sky, as shown in Figure 1. We know there are n flying saucers and each of them occupies the open interval (L_i, R_i) (assume L_i, R_i are integers). To destroy those flying saucers, we are going to fire the laser canon at some locations. If the laser canon is fired at position x to the sky, it will destroy all the saucers that intersects with this vertical line, i.e., all the flying saucers with $x \in (L_i, R_i)$ will be destroyed, as illustrated in Figure 1. However, firing the laser canon is expensive so we want to find a way to destroy all the flying saucers using as few laser canons as possible.

Mathematically, given n intervals $\{(L_i, R_i) \mid i = 1, \ldots, n\}$, our goal is to find a minimum set of numbers $X = \{x_1, \ldots, x_k\}$ such that for every interval i, there is at least one x_j in X contained in the interval $(L_i < x_j < R_i)$. Give a ~~linear time~~ $O(n\log n)$ algorithm to solve this problem, and prove the correctness of your algorithm.

Algo

S: set of all saucers
Sort S by increasing R_i
while S isn't empty:
choose saucer that has smallest R_i
add $R_i - \varepsilon$ to X
delete all saucers that intersect with $R_i - \varepsilon$ from S
X is the solution set

Proof

Suppose Y is optimal solution set. $y_1 \le x_1$ because shooting greater than x_1 means missing (L_1, R_1) ← first one in sorted order. Given that $x_{i-1} \ge y_{i-1}$, $x_i \ge y_i$ because the algo doesn't have to choose any x_i less than y_i (in the worst case it can just choose y_i). ∴ Always, $x_i \ge y_i$.

$k := |X|$, $m := |Y|$. If $k > m$, then $x_m \ge y_m$. So, $x_k > y_m$. Because of $x_m \ge y_m$, S should be empty after x_m. The algo stops when S is empty so $x_k \notin X$ — a contradiction. ∴ $k = m$.