CS180-Spring20 Midterm

[redacted]

TOTAL POINTS

99 / 100

QUESTION 1

1 T of F 20 / 20

✓ **- 0 pts Correct**

- 10 pts No justification for any of the answers

- 5 pts Part justification

- 5 pts a wrong

- 5 pts b wrong

- 5 pts c wrong

- 5 pts d wrong

- 7.5 pts missing 3 justifications

- 2.5 pts b justification is wrong

QUESTION 2

2 Order of growth 10 / 10

- 1 pts 1 incorrect pair.

- 2 pts 2 incorrect pairs.

- 3 pts 3 incorrect pairs.

- 4 pts 4 incorrect pairs.

- 5 pts 5 incorrect pairs.

- 6 pts 6 incorrect pairs.

- 7 pts 7 incorrect pairs.

- 8 pts 8 incorrect pairs.

- 9 pts 9 incorrect pairs.

✓ **- 0 pts Correct**

- 10 pts 10 incorrect pairs.

QUESTION 3

3 DAG 20 / 20

✓ **- 0 pts Correct**

- 20 pts No answer

- 1 pts Hard to read

- 2 pts Correct idea, but needs a more formal algorithm description

- 2 pts No explanation for algorithm's correctness

- 1 pts Needs better explanation for algorithm's correctness. Need to justify that your algorithm returns true if and only if G contains such a path.

- 8 pts Algorithm produces correct output, but with incorrect time complexity.

- 10 pts Incorrect, but a complete answer is given

- 1 pts Algorithm description needs some additional detail.

QUESTION 4

4 Array 20 / 20

✓ **- 0 pts Correct**

- 2 pts no proof of correctness or explanation

QUESTION 5

5 Flying saucers 29 / 30

✓ **- 0 pts Correct**

- 10 pts no correctness proof

- 2 pts hard to read

- 5 pts no time complexity proof

- 2 pts incomplete correctness proof

- 5 pts wrong time complexity proof

- 10 pts incorrect algorithm

- 30 pts no answer

- 5 pts incomplete algorithm

- 0 pts late submission

- 5 pts Not a regious description (or hard to follow)

- 0 pts Click here to replace this description.

- 1 Point adjustment

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.

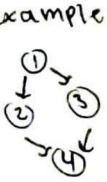
   (b) (5pt) For a DAG G, if there is only one node with no incoming edge, then there exists only one topological ordering.

   (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.

a) True. If removing an edge $e:(u,v)$ disconnects a graph this means e is the only edge connecting the connected component with v and the connected component with v. By definition the DFS will traverse all edges and put the edge in the DFS tree if node v has not yet been visited. Since e is the only edge to v from the connected component of u, it must be in the DFS tree as v could not have been visited (or vice versa if DFS visited v first) before

b) False. counter example:

[DAG] 

only one node has no incoming edge (1).

topo order 1 

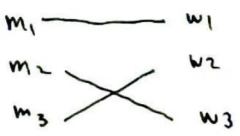
topo order 2 

As seen there exisits more than one topological ordering for the DAG that only has node 1 w/ no incoming edges.

c) False. counterexample: If all other men and women had m_1 and w_1 as their lowest preference.

let m pref:
- m_1 : $w_3 > w_2 > w_1$
- m_2 : $w_3 > w_2 > w_1$
- m_3 : $w_2 > w_3 > w_1$

and w pref:
- w_1 : $m_3 > m_2 > m_1$
- w_2 : $m_2, m_3 > m_1$
- w_3 : $m_3 > m_2 > m_1$

clearly 

is stable

As no other $m \neq m_1$ prefers w_1 over any other $w \neq w_1$ and no other $w \neq w_1$ prefers m_1 over any $m \neq m_1$ So none will break their current matchings and this is a stable matching with (m_1, w_1).

d) True. Since this is a DAG, there are no cycles thus if a node is a leaf, after running DFS, there can be no back edges to ancestors as that would create cycles. There is also no edges to other branches if u is the first leaf node in the DFS tree b/c if there was, then DFS algorithm would have traversed that edge (since u is the first leaf) but if DFS traversed that edge, u will no longer be a leaf and we have a contradiction. Therefore, if we run DFS on a DAG and node u is the first leaf in the DFS tree, then u has no outgoing edge.

1 T of F 20 / 20

✓ - **0 pts** **Correct**

- **10 pts** No justification for any of the answers
- **5 pts** Part justification
- **5 pts** a wrong
- **5 pts** b wrong
- **5 pts** c wrong
- **5 pts** d wrong
- **7.5 pts** missing 3 justifications
- **2.5 pts** b justification is wrong

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 $O(g(n))$.

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

① $f_5(n) = 2^{0.8 \log n} \rightarrow n^{0.8} = O(n)$

② $f_2(n) = n(\log n)^{100} \rightarrow n(\log n)^{100} = \Omega(n)$ so $n^{0.8} = O(n(\log n)^{100})$

③ $f_1(n) = 3n^3$; $\lim_{n \to \infty} \frac{3n^3}{n(\log n)^{100}} = \infty > 0$ so $f_2(n) = O(f_1(n))$

④ $f_4(n) = 2^{\sqrt{n}}$ // exponential rates grows faster than polynomial so $f_1(n) = O(f_4(n))$

⑤ $f_3(n) = 2^{n \log n} \rightarrow n^n$ // $2^{\sqrt{n}} = O(2^n) = O(n^n)$

2 Order of growth 10 / 10

- **1 pts** 1 incorrect pair.
- **2 pts** 2 incorrect pairs.
- **3 pts** 3 incorrect pairs.
- **4 pts** 4 incorrect pairs.
- **5 pts** 5 incorrect pairs.
- **6 pts** 6 incorrect pairs.
- **7 pts** 7 incorrect pairs.
- **8 pts** 8 incorrect pairs.
- **9 pts** 9 incorrect pairs.

✓ **- 0 pts Correct**

- **10 pts** 10 incorrect pairs.

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.

Run topological sort to get the topological ordering $\Gamma = \{v_1, v_2, \ldots v_n\}$ // Runs in O(m)

```
while (i ≠ n)
    If (vi, vi+1) ∉ E        // E = set of edges; use adjacency matrix so O(1) for each check and there are n < m checks so total → O(m)
        return false
return true
```

Proof of time complexity

Topological sort is given to run in time O(m) and checking that each node is visited exactly once is O(m). Thus the overall time complexity is O(m)

Proof of correctness

By running topological sort on a DAG we get a topological ordering s.t all edges are pointing forward. This means only nodes with lower ordering can have edges to those with higher ordering. However, each node must be visited exactly once, therefore we cannot have multiple edges going to one node (for a single path). Therefore there can only be an edge from the preceding node if we are to have a path that visits every node exactly once.

If we apply this to every node there exists a path $P: v_1 \to v_2 \to v_3 \to \ldots \to v_n$ such that each node is visited exactly once.

- Additionally it is clear that the edge had to exist from the node that directly preceded because if you chose any other preceeding edge (ie v_{i-2}), you would have had to 'skip' a node in between (v_{i-1}) which then cannot be reached because you cannot have edges pointing backwards additionally there would be no path that can reach both v_i and v_{i-1} in that case and you would not be able to have a path that visits every node once.
- The algorithm correctly detects that there exists an edge for every v_i, v_{i+1} in order for it to return true and will return false otherwise.

3 DAG 20 / 20

✓ **- 0 pts** **Correct**

- 20 pts No answer

- 1 pts Hard to read

- 2 pts Correct idea, but needs a more formal algorithm description

- 2 pts No explanation for algorithm's correctness

- 1 pts Needs better explanation for algorithm's correctness. Need to justify that your algorithm returns true if and only if G contains such a path.

- 8 pts Algorithm produces correct output, but with incorrect time complexity.

- 10 pts Incorrect, but a complete answer is given

- 1 pts Algorithm description needs some additional detail.

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.

Use a variation of binary search: Find_index(A, 1, n)

```
Find_index(A, l, u):
    i = l + ⌊(u - l)/2⌋
    if l = u and A[l] ≠ l : return false
    if A[i] = i : return true (and i if you want)
    else if A[i] < i : return Find_index(A, i+1, u)
    else if A[i] > i : return Find_index(A, l, i-1)
```

Proof of Time complexity

Each time you are decreasing the search space to half, thus the algorithm is $O(\log n)$ [as the other operations like comparisons and setting i are $O(1)$ time].

Proof of correctness

If $l = u$ that means there is only one element/one element left. If the value of that element does not equal its index there is obviously no elements left to make it true. If $A[i] = i$ it is, by definition of the problem statement, true.

Since the list is sorted in increasing order and elements are distinct $\Rightarrow$ A is strictly increasing

Therefore, it is impossible for $A[j] = j$ for $j > i$ if $A[i] > i$

↳ i.e. $A[5] = 7 > 5 \Rightarrow A[6]$ must be at least 8.

So we can ignore all $A[j]$ s.t. $j > i$ and only search indexes less than i.

Similarly it is impossible for $A[j] = j$ for $j < i$ if $A[i] < i$

↳ i.e. $A[2] = 0 < 2$ then $A[1]$ is at most -1.

So we can ignore all $A[j]$ s.t $j < i$ and only search indices greater than i.

By narrowing our search space in half we will either:

① Find $A[i] = i$ → ret true

② Narrow the search space until there is only one item
   a) value = index → true
   b) value ≠ index → false

Therefore, we are able to find whether there is an index i with $A[i] = i$ at the end of this algorithm.

4 Array 20 / 20

✓ - 0 pts Correct

- 2 pts no proof of correctness or explanation

Question 5

Sort by smallest rhs value and put in a linked list: R_list.

Sort by smallest lhs value and put in a queue : L_queue

Initialize laser = 0

u = R_list → head , v = L_queue.pop()

remove (R_list → head)

while (R_list is not empty)
    laser = laser + 1
    while (v overlaps with u) // just do some constant comparisons for LHS/RHS
        remove v from R_list (if v ≠ u)
        v = L_queue.pop()
        x_{laser} = some x in the smallest overlap range btwn u and all v seen in this loop
    u = R_list → head
    remove (R_list → head)

↑ just store current min range seen so far

Proof of Time complexity

Sorting R_list and L_list takes O(nlog n). Then each element is compared once for overlap (and is removed from the list) so the whole entire part of the while loop runs in O(n) time. [If we keep an auxilary array with pointers to locations in R_list - indexed by the saucer # - we can remove/find the item to remove in R_list in O(1) time] The sorted L_queue also helps to find items that overlap in O(1) time. Therefore the overall time complexity is dominated by sorting → O(nlog n) time.

Proof of correctness:

Since we need to shoot all the saucers, by starting with the saucer with the smallest right hand side we know we need to fire at least one laser at that point to destroy it. Therefore we find all the other saucers in that range. We just have to consider that the Lhs of those saucers appear at times before the rhs of the saucer we currently are looking at. As long as we fire the laser at some x that is in the smallest overlap range between u and all v's it overlaps with, it should destroy all v's that overlap with u (as there is no way it can miss because each v's RHS is by definition larger than u's RHS) (except when u=v, then we ignore). Then we move on to the next element in R_list that has not been removed, and repeat. At the end of the algorithm we obtain the min # of lasers and set x that is minimized because at each selection x_i we had to have at least one laser there to destroy u (and consequently those it overlaps) as the lasers only travel in straight (vertical) paths. My implementation scanned from left to right for finding positions of the lasers but an implementation from right to left should also work and possibly give a different minimized answer.

5 Flying saucers 29 / 30

✓ - 0 pts **Correct**

- 10 pts no correctness proof
- 2 pts hard to read
- 5 pts no time complexity proof
- 2 pts incomplete correctness proof
- 5 pts wrong time complexity proof
- 10 pts incorrect algorithm
- 30 pts no answer
- 5 pts incomplete algorithm
- 0 pts late submission
- 5 pts Not a regious description (or hard to follow)
- 0 pts Click here to replace this description.

- 1 **Point adjustment**