CS180 Midterm Exam Solutions

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.

   Solution:

   (a) True. All the non-tree edges are back-edge in DFS which means the non-tree edges are involved in some cycle. Moreover, since e is a cut for the graph G, we know DFS will have to pass it when traversing the graph.

   (b) False. Example constraints (a,b) (a,c), (c,d), (b,d). Clearly, there could be 2 ordering. (a,b,c,d) and (a,c,b,d).

   (c) False. As an example, consider the following ranking: m_1 and m_2 prefer w_2 to w_1. w_1 and w_2 both prefer m_2 to m_1. A stable match will then be (w_1, m_1)&(w_2, m_2).

   (d) True. u can only have back edge to its ancestor, and if back edge exists the graph will not be a DAG. Therefore u has no outgoing edge.

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 $O(g(n))$.

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

   Solution: f_5, f_2, f_1, f_4, f_3 (totally 10 pairs, each pair 1 point)

   $f_5(n) = 2^{0.8 \log n} = 2^{0.8 \frac{\log_2 n}{\log_2 e}} = n^{\frac{0.8}{\log_2 e}} = O(n(\log n)^{100}) = O(f_2(n))$ (since $\frac{0.8}{\log_2 e} < 1$).

   Let $z = \log n$, by (2.9), we have $(\log n)^{100} = z^{100} = O(e^z) = O(n)$. Hence, we can get $f_2(n) = n \cdot (\log n)^{100} = O(3n^2 \cdot n) = O(f_1(n))$.

   Let $z = \sqrt{n}$, $f_1(z) = 3z^6$, $f_4(z) = 2^z$. By (2.9), we have $f_1(n) = O(f_4(n))$.

   Since $\sqrt{n} = O(n \log n)$, we have $f_4(n) = O(f_3(n))$.

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.

Solution: The algorithm is given by the following pseudo-code.

```
Obtain a topological ordering of the vertices of G as u_1,...,u_n, using topological sort.
For i = 1,...,n-1:
    If (u_i, u_{i+1}) ∉ E(G):
        return no
return yes
```

Note: there are valid proofs other than the one given here. Also, a less rigorous argument would suffice to get full credit, since we only ask for a justification.

Proof of correctness: Since G is a DAG, the first step of the algorithm always returns a valid topological ordering, $u_1,\ldots,u_n$. Since $u_1,\ldots,u_n$ is a topological ordering, we know that if $(u_i,u_j) \in E(G)$, then $i < j$. Thus, for any path $p = (u_{t_1},\ldots,u_{t_k})$ of length k, we must have $t_1 < t_2 < \cdots < t_k$. Thus, a path p contains every vertex if and only if $p = (u_1,\ldots,u_n)$. Therefore, G has a path containing every vertex if and only if $(u_i,u_{i+1}) \in E(G)$ for every $i \in \{1,\ldots,n-1\}$.

Runtime analysis: For a DAG G on n nodes and m edges, topological sort runs in time $O(n+m)$. Our algorithm first runs topological sort and then checks whether n different edges exist in the graph. Since $m > n$, the total runtime of our algorithm is thus $O(n+m)+O(n) = O(m)$.

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.

Solution:

```
find_index(int start, int end, int A[])
  if end<start:
    return 0
  mid = ⌊(start + end)/2⌋
  if mid - A[mid]=0:
    return 1
  if mid - A[mid]<0:
    return find_ind(start, mid-1, A[])
  if mid - A[mid]>0:
    return find_ind(start+1, mid, A[])
```

```
find_index(int start, int end, int A[])
  if end<start:
    return 0
  while (end >= start):
    mid = ⌊(start + end)/2⌋
    if mid - A[mid]=0:
      return 1
    if mid - A[mid]<0:
      end ← mid-1
    if mid - A[mid]>0:
      start ← mid+1
  return 0
```

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 algorithm to solve this problem, and prove the correctness of your algorithm.

Algorithm:

```
Define K = [(L_i, R_i)]_{i=1,...,n}
K_s = sorted(K, key = R_i, ascending = True)
While not K.empty():
  L_i, R_i=K[0]
  Fire a laser beam at R_i - 0.5
  Remove the destroyed saucers, update K to be the suviving saucers.
End
```

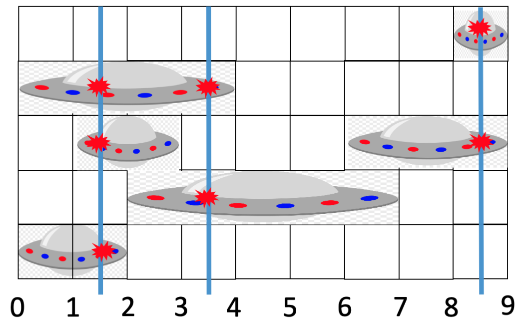

Figure 1: In this example, there are 6 flying saucers with $(L_1, R_1) = (0,2)$, $(L_2, R_2) = (2,7)$, $(L_3, R_3) = (1,3)$, $(L_4, R_4) = (6,9)$, $(L_5, R_5) = (0,4)$, $(L_6, R_6) = (8,9)$. We need at least 3 laser canons to destroy all of them, and $1.5, 3.5, 8.5$ is a set of valid positions of these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

Time complexity: Sorting takes $\mathcal{O}(n \log n)$, the while loop takes $\mathcal{O}(n)$.

Reasoning: Suppose otherwise, there is a better solution S^* that fires fewer beams. Consider the positions of the left-most beams of both algorithms, x and x^*; we must have $x > x^* - 0.5$, otherwise, x^* will not be able to destroy the left-most saucer (located at $K[0]$). After that, the remaining saucers of our algorithm is a subset of optimal solution. But optimal solution uses fewer beams than our solution, which leads to a contradiction.