CS180 Midterm Exam Solutions

1. (20 pt) Determine whether the following statements are **true or false**.

   (a) (5 pt) $2^n = \Theta(3^n)$

   (b) (5 pt) Suppose G is a connected, undirected graph. If removing an edge disconnects the graph, then the edge is a tree edge in the DFS tree.

   (c) (5 pt) Suppose G is a DAG and s is the first node in the topological ordering, t is the last node in the topological ordering, then there is always a path from s to t.

   (d) (5 pt) For any undirected connected graph G, there exists at least two nodes such that removing the nodes won't disconnect the graph.

   Solutions:

   (a) False

   (b) True

   (c) False

   (d) True (we can build a spanning tree using BFS/DFS, then removing leaf nodes won't disconnect the graph. And each tree has at least 2 leaf nodes).

2. (20 pt) As we learned in the class, a min-heap is useful when we are interested in the minimum value of set of numbers. Now, instead of the min, we are interesting in the K-th smallest value.

   (a) (10 pt) Please design a data structure to allow the following two operations: `push` (insert a new number) and `find_Kmin` (return the value of the K-th smallest number without removing it). Both operations should be done in $O(\log K)$ time.

   (b) (10 pt) In addition to `push` and `find_Kmin`, now we also want to support the `pop` operation to return and remove the K-th smallest element. Please design a data structure to support these three operations, and each operation should take $O(\log n)$ time with n being the size of the current set.

Solution: (a) use a max heap A to maintain the smallest K elements. When pushing , compare with the max of A to determine whether we want to add the new element to the heap. (b) In addition to A, also using another min heap B to store the rest of the elements. When popping an element, moving the max element in B to A.

3. (20 pt) For a stable marriage problem with n men and n women, let m_1, m_2 be two of the men, w_1, w_2 be two of the women. Suppose m_1's preference list is $w_1 > w_2 > \ldots$; m_2's preference list is $w_2 > w_1 > \ldots$; w_1's preference list is $m_2 > m_1 > \ldots$; w_2's preference list is $m_1 > m_2 > \ldots$ (we only know the favorite and the second favorite of each of these four persons).

   (a) (10 pt) Show that in every stable matching, m_1, m_2 are matched to w_1, w_2 (which means we either have $(w_1, m_1), (w_2, m_2)$ or $(w_1, m_2), (w_2, m_1)$ in a stable matching).

   (b) (10 pt) For any even n, construct an instance of stable matching problem (i.e., provide a set of preferences) with at least $2^{n/2}$ stable matchings. Justify your claim.

Solution: (a) Contradiction: If any ends up paired with someone else outside of the small group, they'd want to swap back into the group, and so would the other gender that also ends up outside of the group.

(b) repeat the same pattern for $w_i, w_{i+1}, m_i, m_{i+1}$ for $i = 1, 3, 5, \ldots, n-1$, where everyone in each of these groups prefers each other first, in the same way as in the statement.
I.e. m_{2i+1} prefers $w_{2i+1} > w_{2i+2} > \ldots$; m_{2i+2} prefers $w_{2i+2} > w_{2i+1} > \ldots$; w_{2i+1} prefers $m_{2i+2} > m_{2i+1} > \ldots$; w_{2i+2} prefers $m_{2i+1} > m_{2i+2} > \ldots$.
For every 2 men out of the n, there are 2 independent possibilities, for $2^{n/2}$ total possibilities.

4. (20 pt) Indiana Jones is stuck in the Emperor's Tomb and needs to find a way out. In front of him, there are n consecutive doors, each door is behind the previous, and he needs to open all the doors to get out. In the control room, there are n switches where each switch connects to a different door but he doesn't know which switch controls which door. Each switch can either be in an up or down position. but unfortunately, for some doors the "up" position will open the door while for others the "down" position will open the door. He needs to place all the n switches in the correct configuration to open all the doors. In order to find the correct configuration, each time Indiana Jones can walk to the control room, set the switches to any configuration, and walk back to check which is the first closed door (since the doors are consecutive, he cannot observe the status of doors behind the first closed one). Design an algorithm to find the correct configuration to open all the doors within $O(n \log n)$ trials.

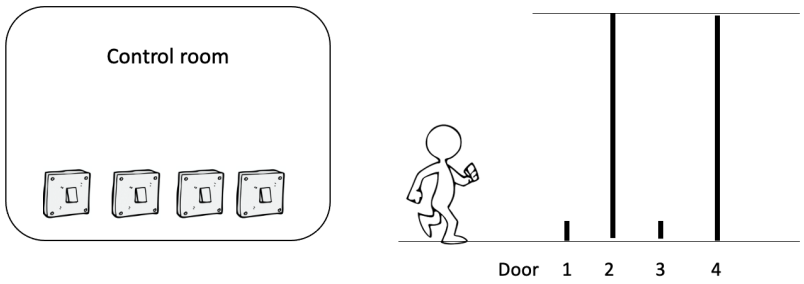

Figure 1: Illustration of Problem 3. In this case there are $n = 4$ doors, and 4 switches controlling those doors. He can only see the state of doors 1 and 2, not 3 or 4

Solution: Find the controller for each door one-by-one. For each door (say door i), use binary search: divide the switches into 2 groups A, B; try two configurations (a) Up for all the switches (b) Up for the first group and down for the second group. If the first door is opened for both cases then the switch for door i should be in group A; otherwise it should be in group B. We can thus reduce the candidate set by half, and doing this recursively can find the switch for door i in $O(\log n)$ trials. There are totally n doors so $O(n \log n)$ trials.

5. (20 pt) There's a 1D segment and you want to travel from west-most point (s) to the east-most point (t). There are n teleporters on this 1D segment and each teleporter has two endpoints. Whenever you reach one endpoint, it will teleport you to another endpoint (it may transport you from east to west or west to east, depending on which endpoint you reach). All the endpoints are located between s and t and none of the endpoints are located at the same position. We assume you must always move **eastward** on the segment.

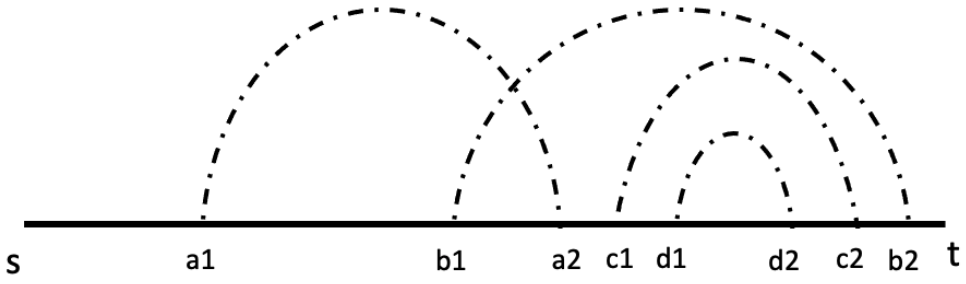

Figure 2: An example for Problem 5. We are given an input with 4 teleporters with endpoints a_1, a_2; b_1, b_2; c_1, c_2, d_1, d_2. Starting from s, you should keep moving to the right, so the path will be s (walk to) a_1 (teleport to) a_2 (walk to) c_1 (teleport to) c_2 (walk to) b_2 (teleport to) b_1 (walk to) a_2 (teleport to) a_1 (walk to) b_1 (teleport to) b_2 (walk to) t, and the path has reward t since you are teleported 5 times.

(a) Prove that no matter how those teleporters are placed, you will always reach the t. (Hint: you can view this as a graph problem, where each segment is a node, and each one has an edge pointing to the next segment)

(b) Now you are allowed to add another set of K teleporters (each with two endpoints). How could you place those endpoints to *maximize* the number of times you get teleported in your trip from left-most point to right-most point? Given the input of the locations of existing teleporters and the number K, design an algorithm to solve this problem in $O(n \log n)$. (You can assume $K < n$.)

Solution:

(a) If we view each segment as a node, then this will be a graph with $2n + 1$ nodes. Let s denote the first segment and t denote the last segment, then we want to know whether there's a path from s to t. In this graph, every node has an incoming and an outgoing edge, except s, t where s has only one outgoing edge and t has one incoming edge. We start from s and keep going to the next node. We can't reach the same node twice since each node has only one incoming edge. Therefore the path will end at t which has no outgoing edge.

(b) The graph we construct to represent the teleportation path is composed of a path from s to t, and possibly several loops. Adding an teleporter can merge two loops or merge a loop into the path. Therefore, we can greedily merge the largest loop into the path when adding each new teleporter. (After all loops are merged you can just add a teleporter with both endpoints on segment. This will create a self-contained segment which is a single disconnected node with a self loop in our directed graph, but you can merge that back into the our path with another teleporter just like how we merged previous loops. In practice this means adding (x1, y1, x2, y2) on any segment for every 2 leftover teleporters, and (x1, x2) if there's an odd one left out.) Solving this problem only requires finding all the disconnected components and connecting them all. Connecting the components is a constant time operation (it can be done on any arbitrary part of the s-t path and the loop), but finding which elements are disconnected (and how big each loop is) requires traversal over all the nodes, which is $O(n)$. We loosened the requirement later to $O(n \log n)$ because creating the graph itself might require

some type of sorting, depending on how the input is presented in the first place. We also later added that $K < n$.

(c) GRADER NOTES: We didn't precisely define exactly how the input data is presented to them, what the format / type of data is, etc. So I would let them define it however they want. Just as long as their algorithm and analysis gets the general idea and doesn't have any major bugs, I would give them mostly full credit. Also keep in mind that the wording for 5b seemed to confuse some people, so I would be forgiving in taking off points for misunderstandings there as long as they give reasonable algorithms that could be extended to this case. Also figuring out how to add teleporters once you've merged all the loops is a little trickier than we originally intended, so I would only deduct a small amount if they didn't manage to figure that out. Merging the loops is the most important concept here.