Fall 2018, CMPSC 465: Take-home Exam 2 solutions.

---

1. Devise an $O(n \log n)$ algorithm for the following problem: Given n points in the plane, find the pair with the shortest distance between them.
   Solution:
   This is a very challenging computational geometry problem. In the exam instructions, I mention that you are only supposed to refer to the three class textbooks, and that accessing online resources is not permitted. I did not expect anybody in the class to solve this problem, without referring to the Goodrich-Tamassia textbook (see Section 22.4). However, nearly all submissions have solutions including some running time analysis! I will only give full credit to honest and clear (but incomplete) attempts, or complete solutions that mention the corresponding section of the Goodrich-Tamassia book. I will deduct points for unclear submissions that mention sorting X and Y coordinates, and then say, hence proved.

2. A mother vertex in a directed graph $G = (V, E)$ is a vertex v such that all other vertices in G can be reached by a directed path from v.
   (a) Give an $O(n + m)$ algorithm to test whether a given vertex v is a mother of G.
   Solution:
   Perform a BFS with v as the source and check if all other vertices are reachable.
   (b) Give an $O(n + m)$ algorithm to test whether graph G contains a mother vertex.
   Solution:
   Perform an SCC decomposition of the graph and check if there a unique source SCC. All vertices in this unique source SCC can be mother vertices.

3. In a **side-scrolling video game**, a character moves through an environment from, say, left-to-right, while encountering obstacles, attackers, and prizes. The goal is to avoid or destroy the obstacles, defeat or avoid the attackers, and collect as many prizes as possible while moving from a starting position to an ending position. We can model such a game with a graph, G, where each vertex is a game position, given as an (x, y) point in the plane, and two such vertices, v and w, are connected by an edge, given as a straight line segment, if there is a single movement that connects v and w. Furthermore, we can define the cost, $c(e)$, of an edge to be a combination of the time, health points, prizes, etc., that it costs our character to move along the edge e (where earning a prize on this edge would be modeled as a negative term in this cost). A path, P, in G is **monotone** if traversing P involves a continuous sequence of left-to-right movements, with no right-to-left moves. Thus, we can model an optimal solution to such a side-scrolling computer game in terms of finding a minimum-cost monotone path in the graph, G, that represents this game. Describe and analyze an efficient algorithm for finding a minimum-cost monotone path in such a graph, G.
   Solution:
   Define a monotone graph G_M to be a graph created by combining all edges that appear in monotone paths of G. G_M will be acyclic, because right-to-left moves are disallowed in a monotone path, and two monotone paths cannot combine to create a cycle (we cannot have edges between two vertices u and v such that the X coordinate of vertex u is greater than the X coordinate of v). The edge weights could be negative. Since G_M is a DAG, we can apply the *shortest paths in DAGs* algorithm from a source vertex. This is a linear time algorithm. The minimum cost monotone path representing the game will be the minimum cost path among all

possible paths to sink vertices in this graph.

4. Exercise 3.20 from the textbook.
   Solution:
   Assume that the old labels are stored in an array l, and that the new labels will be stored in l_{new}. For each vertex v in the tree, we need to determine its $l(v)^{\text{th}}$ ancestor. Once we know the $l(v)^{\text{th}}$ ancestor, we can look up the label of the ancestor in the old label array, and set the new label of v. So how do we determine k^{th} ancestor of a vertex? This depends on the value of k and the depth of v in the tree (or the distance of v from the root node). If $k \geq depth(v)$, then the kth ancestor is going to be the root node r. If $k < depth(v)$, then the kth ancestor is some vertex from the path from v to r. Because this is a tree, there can only be one path from v to r. If all the vertices along the path from r to v are stored in an array P_v, with $P_v[0] = r$ and $P_v[depth(v)] = v$, then the kth ancestor is $P_v[depth(v) - k]$. In the tree, we will have leaf nodes and non-leaf nodes. Observe that it suffices to create these path arrays for the tree nodes. The non-leaf nodes can reuse the path arrays of the leaf nodes, because the jth ancestor of a node v would be the $(j-1)^{th}$ ancestor of its parent. Putting all these ideas together, we do a depth-first search from tree, keep track of the path arrays, update the labels in a top-to-bottom fashion based on the current label of the vertex.

5. Exercise 4.22 from the textbook.
   Solution:
   Parts (a) and (b) are similar and easy to show. For part (c), perform binary search on r^* and use Bellman-Ford to detect negative cycles. The running time is $O(\log(\frac{R}{\epsilon})nm)$.