CS 6363.003 Homework 4

Due Sunday April 18th on eLearning

Please answer the following **4** questions, some of which have multiple parts.

1. Stick Clash is puzzle game for mobile devices by Doan Thanh where you control a stick figure knight on a quest to save their captive friend. (The following description of the game is largely based on reality but many details have been changed by Kyle.) Along their way, the knight may recruit friendly soldiers who will join the knight's party and help them on their quest; however, the knight must sometimes battle with the enemy soldiers before they can be convinced to befriend the knight and join the party.

   For the purposes of this problem, we'll consider the following variant of Stick Clash. A single instance of the game takes place on a directed graph $G = (V, E)$. The knight and their initial party start their quest on a vertex s, and the knight's captive friend waits at another vertex $t \neq s$. In each turn of the game, you may move the knight's party from their current vertex u to any vertex v where $u \to v \in E$. You win the game as soon as the knight's party reaches vertex t. The *strength* of the knight's party at any point in time is represented by a real number x which is initially set to 20.

   To make the game challenging, each vertex $v \notin \{s, t\}$ contains *at most one* of several features. The effects of a feature take place only the first time (if ever) the knight's party reaches the vertex v. The possible features are as follows:

   - ***A group of friendly soldiers of strength $g(v)$***: The soldiers immediately join the knight's party, and the party's strength increases as $x \leftarrow x + g(v)$.
   - ***A group of enemy soldiers of strength $b(v)$***: The knight's party fights the enemy soldiers. If $x \leq b(v)$, the knight's party loses the battle and you immediately lose the game. Otherwise, the knight recruits the enemy soldiers and the party's strength increases as $x \leftarrow x + b(v)$. For example, if $x = 30$ and $b(v) = 20$, then the party wins the battle and their strength increases to $50 = 30 + 20$.
   - ***A spike trap of harm $h(v)$***: The knight's party takes damage and their strength decreases as $x \leftarrow x - h(v)$. If $x \leq 0$ after the knight's party takes damage, you immediately lose the game.
   - ***A blessed shield***: The knight's party is blessed by the shield and their strength *doubles* as $x \leftarrow 2 \cdot x$. The knight's party may be blessed more than once during their quest if they reach more than one vertex featuring shields.
   - ***Some cursed bombs***: The knight's party suffers a horrible curse and their strength *halves* as $x \leftarrow x/2$. The knight's party may be cursed more than once during their quest if they reach more than one vertex featuring bombs.

   In general, Kyle does not think there is an efficient way to determine if a given instance of the game can be won. However, certain assumptions on the input graph G may help.

Describe and analyze an efficient algorithm to determine whether or not a given instance of Stick Clash can be won, *assuming the input graph $G = (V, E)$ is a DAG*. You may assume comparisons and basic arithmetic operations, including division, can be done in constant time each.

2. Throughout our lecture on minimum spanning trees, we assumed that no two edges in the input graph have equal weights, a condition that implies the minimum spanning tree is unique. In fact, the minimum spanning tree may be unique even if some pairs of edges have equal weights.

   (a) Describe an edge-weighted undirected graph that has a unique minimum spanning tree, even though two edges have equal weights.

   (b) Let G be an arbitrary edge-weighted undirected graph and F be a subgraph of some minimum spanning tree of G. Let e be an arbitrary safe edge with respect to G and F. Prove the following extension of Erickson's Lemma 7.2:

   Claim. *There exists a minimum spanning tree T of G such that $(F \cup \{e\}) \subseteq T$. Further, every minimum spanning tree $T \supset F$ contains e if there exists a component of F such that e is the only minimum-weight edge leaving that component.*

   [Hint: Modify the proof of Lemma 7.2.]

   (c) Describe and analyze an algorithm to determine whether or not a given edge-weight connected undirected graph has a unique minimum spanning tree. *[Hint: Modify Kruskal's algorithm.]*

3. Suppose we are given a directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$ and two vertices s and t. You may assume G has no negative weight cycles.

   (a) Describe and analyze an algorithm to find the shortest path from s to t when exactly one edge in G has negative weight. *[Hint: Modify Dijkstra's algorithm. Or don't.]*

   (b) Describe and analyze an algorithm to find the shortest path from s to t when exactly k edges in G have negative weight. Any $O(f(k)E \log V)$ time algorithm where f is a function of k is worth full credit, but an $O(kE \log V)$ time algorithm may be faster and easier to analyze than those with worse dependency on k. *[Hint: Modify Bellman-Ford so it sometimes calls a variant of Dijkstra's algorithm in the subgraph of non-negative weight edges.]*

4. Let $G = (V, E)$ be a directed graph with edge weights $w : E \to R$; edge weights could be positive, negative, or zero, but you may assume there are no negative weight cycles.

   (a) Let $v \in V$ be an arbitrary vertex. Describe and analyze an algorithm that constructs a directed graph $G' = (V \setminus \{v\}, E')$ with weighted edges such that the shortest path distance between any two vertices in G' is equal to the shortest path distance between the same two vertices in G. Your algorithm should run in $O(V^2)$ time.

(b) Now suppose we have already computed all shortest path distances in G'. Describe and analyze an algorithm to compute the shortest path distances in the original graph G from v to every other vertex, and from every other vertex to v, all in $O(V^2)$ time.

(c) Combine parts (a) and (b) to describe and analyze another all-pairs shortest paths algorithm that runs in $O(V^3)$ time. (The resulting algorithm is *almost* the same as Floyd-Warshall.)