# Problem Set 7

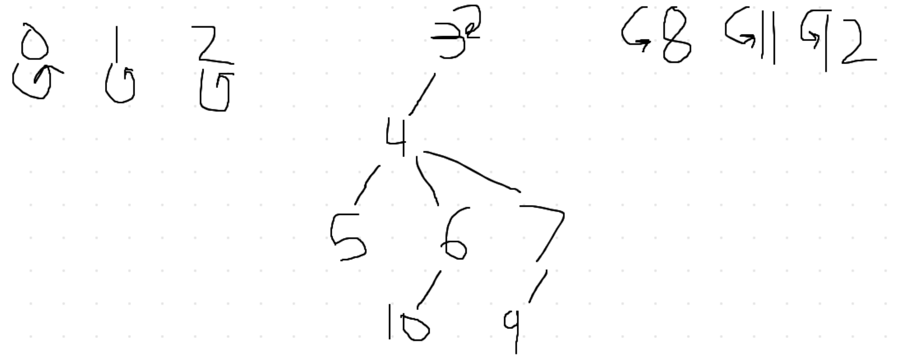
**All parts are due Thursday, November 1 at 11PM**.

**Name:** Kevin Jiang

**Collaborators:** Suraj Srinivasan, Sathwick Karnik, Rajiv Movva

**Problem 7-1.**

**(a)** Topological sort requires the graph to be acyclic, so cycles like (4,7), (7,4) and (7,9), (9,7) require at least one edge to be deleted for each.



**(b)** F = [0, 1, 2, 5, 10, 6, 9, 7, 4, 3, 8, 11, 12]

**(c)** Only edges $(7, 4)$ and $(9, 4)$ need to be deleted.

**Problem 7-2.** Note that this problem is the same as Bellman-Ford's algorithm except that we only need to relax $E$ edges $k$ times since the longest shortest path will have $k$ edges rather than $V - 1$. Initializing the $V$ vertices to infinite distance takes $O(V)$ time. Relaxing $E$ edges $k$ times takes $O(E \cdot k)$ time. Therefore, the overall run time is $O(V + E \cdot k)$.

**Problem 7-3.** We create a graph $G = (V, E, w)$ where the vertices are the materials, the edges $(u, v) \in E$ convert material $u$ into $v$, and the weights are $r$. Modify the weights of each edge in our graph to contain $\log(\frac{1}{w})$ rather than $w$. We do $\frac{1}{w}$ so that finding the shortest path from salt to gold will find the maximum conversion factor from salt to gold. We take the $\log$ of each weight to simulate multiplication of mass ratios. Therefore, to find the maximum amount of gold Germoine can make from $s$ grams of salt, we find the shortest path from salt to gold, $\pi$, and calculate $s \cdot e^{w(\pi)}$. We can do this by using version two of Bellman-Ford's algorithm, which has a run time of $O(|V||E|)$. Note that a negative weight cycle in the path from salt to gold in this new graph will lead to infinite gold.

**Problem 7-4.** Note that the map is a directed acyclic graph, $G = (V, E, w)$, where the vertices are the caves, the edges start from a lower cave and point to a higher cave, and the weights of each edge $(u, v) \in E$ are the amount of damage that will be dealt to Zink's sword after fighting in cave $v$ (we assume that the starting vertex is not dangerous). Since the graph is a DAG, we use topological sort relaxation starting at the beginning cave to return a list $L$ of shortest paths to each cave $\in V$. Next, for each vertex $v \in L$, we check if it is at the surface elevation, and if $L - d_v > 0$. If so, then we return True. Otherwise, if no such $v$ exists, then return False.

This algorithm is correct since the directed edges guarantee that Zink and Lelda travel upwards, and the shortest path returns the minimum amount of damage that will be taken by Zink's sword. The runtime is $O(|V| + |E|)$ since it takes $O(|V| + |E|)$ to generate the graph, $O(|V| + |E|)$ to run topological sort relaxation, and $O(|V|)$ to check if an escape is possible (assuming elevation look up is $O(1)$).

**Problem 7-5.** We create a graph $G = (V, E, w)$ where the vertices are the road intersections, the edges are the streets with agents and point in the direction where houses are on the right, and the weight of each edge $(u, v) \in E$ is $a - h$ ($h$ denotes the number of houses on the right). We run the second version of Bellman-Ford to find the shortest path from Jopper's shed to Whike Meeler's house. If the shortest path is $< 0$, then Lee can reach Whikes house with strictly more candy than when she started. If a negative weight cycle exists in the path from the shed to the house, then Lee can get infinite candy, and we return True.

This algorithm is correct since finding the shortest path, $pi$, in this graph will tell us that Lee will gain $-w(\pi)$ candy. Note that between any two adjacent vertices, there will be two edges going between them (AKA a 2 vertex cycle). Moreover, $a - h$ maybe positive or negative since Lee could gain or lose candy. Therefore, $G$ is a directed cyclic graph that may have negative weights. Hence, the most efficient algorithm to deal with cycles and negative weights is Bellman-Ford's, which has a run time of $O(|V||E|)$.

**Problem 7-6.**

**(a)** Finding the maximum weight path, $\pi$, from any vertex in a subset $S \subset V$ to a vertex $t$ is equivalent to finding the maximum path from $t$ to a vertex in subset $S$. We generate a new graph $G' = (V, E', -w)$ where each edge weight, $w(u, v)$, is replaced with its negative version, $-w(u, v)$, and each edge $(u, v) \in E$ is reversed to $(v, u) \in E'$. In $G'$, we find the maximum path from $t$ to $s \in S$ by running the topological sort relaxation algorithm.

This algorithm is correct since finding the minimum path, $\pi$, from $t$ to $s \in S$ of a graph of all negative weights will give us the largest path from $t$ to $s$ when we calculate $-w(\pi)$. Moreover, we can use topological sort relaxation since the directed acyclic graph property is invariant during the transformation. The overall run time is $O(|V| + |E|)$ since it takes $O(|V| + |E|)$ to both create $G'$ and run topological sort relaxation.

**(b)** Given a list of code transformations of the form $(F, o, m)$, we construct a graph $G = (V, E, w)$ by having the vertices be the file names, the edges $(u, v) \in E$ represent the transformation from file $u$ to file $v$, and the weights be the time it takes to convert file $u$ to file $v$. We construct the edges by connecting each input file, $f_i \in F$, to the output file $f_o$ and set each edge weight, $w(f_i, f_o)$, to $m$. To compute the minimum number of milliseconds needed to build target file $t$ from the source files $S$, we run part (a) on the graph $G$ to find the list of maximum-weight paths from $t$ to a source file $S$ and return the maximum value of the set of maximum-weight paths.

Due to the fact that Parry can run code transformations simultaneously, the minimum number of milliseconds needed to build file $t$ is the maximum of the times to go from a source file $s \in S$ to $t$ (this maximum acts as a ”limiting factor”). Therefore, finding the minimum time to build $t$ is just the sum of the ”limiting factors” tracing back to the source files. In other words, we just want the longest path from any source file to $t$.

**(c)** Submit your implementation to `alg.mit.edu/PS7`