Assignment 4 (due Wednesday, March 18th, 6:00pm)

Instructions:

- Hand in your assignment using Crowdmark. Detailed instructions are on the course website.
- Give complete legible solutions to all questions.
- Your answers will be marked for clarity as well as correctness.


You are loading a stream of trucks arriving in a single line into a cargo ship that has two sides (separated by a wall). Each side of the ship can hold maximum weight $M$ (so at most $2M$ in total, but at most $M$ per side). At each step, you can tell a truck to drive onto either the left or right side. In total there are $n$ trucks, with weights $w_1, w_2, \ldots, w_n$. You would like to fit as many trucks as possible onto the ship. You must take a prefix of the trucks (i.e, $w_1, w_2, \ldots, w_i$). If you do not take truck $w_i$, then you cannot take truck $w_{i+1}$. How many trucks can you fit on the ship?

Example. Suppose $M = 50, w_1 = 25, w_2 = 30, w_3 = 10, w_4 = 10, w_5 = 15, w_6 = 7, w_7 = 8$. Then the optimal number of trucks is 6. (Trucks 2-4 can go on the right, yielding a weight of exactly 50, and trucks 1, 5 and 6 on the left, yielding a right side weight of 47. Note trucks 1-6 form a prefix of the input.)

Give a recurrence (specifying all base cases and general case(s), as necessary), and explain what quantity your recurrence represents.

Give pseudocode that solves the problem (clearly showing what order your DP array should be filled in, and how to output the optimal answer).

Provide runtime analysis and justify correctness (optimal substructure).

Hint: a solution with running time $O(nM^2)$ will suffice, but bonus marks are available for solutions that beat this asymptotically.


The input is an unweighted DAG $G = (V, E)$ with at most one path between any pair of vertices. (I.e., for each pair of vertices $u, v$ there is at most one path from $u$ to $v$.)

The desired output is the length of the longest path in the graph.

Provide an efficient algorithm (including pseudocode\footnote{You are allowed to reference graph algorithms that you learned about in class, such as BFS, DFS, SCC, topsort, etc. If you want to modify such an algorithm, you can simply explain the changes you would make. You might wonder how to include precise pseudocode without completely replicating all of the pseudocode for an algorithm you want to modify. As an example of how to do this well, you could explain in words how you would modify BFS() to obtain a new procedure called ModifiedBFS(), and then invoke ModifiedBFS() in your pseudocode. If in doubt, make your solution as clear and understandable as possible.}) to solve this problem.

Provide runtime analysis and justify correctness.

The input is a directed unweighted graph $G = (V, E)$ and particular vertex $s$.

The desired output is the size of the largest strongly connected component (SCC) that is reachable from $s$.

Provide an efficient algorithm (including pseudocode\textsuperscript{1}) to solve the problem.

Provide runtime analysis and justify correctness.


Consider an undirected weighted graph $G = (V, E)$, a source node $s$ and a target node $t$. Let $P$ be any path from $s$ to $t$. Consider the largest edge weight on this path. We call this the bottleneck edge. In this problem, we ultimately want to determine the weight of the minimum bottleneck edge on any path from $s$ to $t$. We will do this in two parts.

(a) [5 marks] Suppose you are given, as inputs, $G$, $s$, $t$ and a weight limit $L$.

Provide an efficient algorithm (including pseudocode\textsuperscript{1}) to determine whether there exists a path from $s$ to $t$ that uses only edges with weight $\leq L$. If such a path exists, return true. Otherwise, return false.

Provide runtime analysis and justify correctness.

(b) [10 marks] Suppose you are given, as inputs, $G$, $s$ and $t$.

Provide an efficient algorithm (including pseudocode\textsuperscript{1}) to determine the weight of the minimum bottleneck edge on any path from $s$ to $t$. (I.e., the minimum, over all paths $P$ from $s$ to $t$, of the weight of $P$’s bottleneck edge.) Return this minimum weight.

Provide runtime analysis and justify correctness.

Hint: try to reduce this optimization problem to the decision problem in (a).


The input is an undirected weighted graph $G = (V, E)$ and a source node $s$. The edge weights in this graph are restricted: Every edge has weight 1 or 3. Let $n = |V|$, $m = |E|$, and $V = (v_1, v_2, ..., v_n)$.

The desired output is an array $d[1..n]$, where $d[i]$ contains the length of the shortest path from $s$ to $v_i$. (In other words, we want to solve the single-source shortest-path problem.)

Provide an $O(n + m)$ algorithm (including pseudocode\textsuperscript{1}) to solve this problem.

Provide runtime analysis and justify correctness.