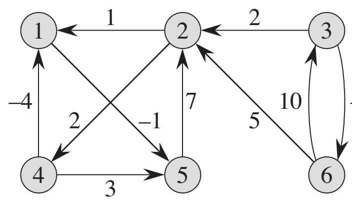
Problem Set 11

Data Structures and Algorithms, Fall 2020

Due: Dec 10, in class.

Problem 1

(a) Run the Floyd-Warshall algorithm on the following weighted, directed graph. Show the $dist$ values for all pairs of nodes after each iteration of the outer-most loop.



(b) How to use the Floyd-Warshall algorithm's output to detect the presence of a negative-weight cycle?

Problem 2

(a) Use Johnson's algorithm to find the shortest paths between all pairs of vertices in the graph used in Problem 1.(a), visualize algorithm's execution via a figure similar to Figure 25.6 in the CLRS textbook.

(b) Suppose that we run Johnson's algorithm on a directed graph G with weight function w. Assume G does not contain negative-weight cycles. Show that if G contains a 0-weight cycle c, then $\hat{w}(u, v) = 0$ for every edge (u, v) in c.

Problem 3

Suppose that we wish to maintain the transitive closure of a directed graph $G = (V, E)$ as we insert edges into E. That is, after each edge has been inserted, we want to update the transitive closure of the edges inserted so far. Assume that the graph G has no edges initially and that we represent the transitive closure as a boolean matrix.

(a) Show how to update the transitive closure of a graph $G = (V, E)$ in $O(|V|^2)$ time when a new edge is added to G.

(b) Give an example of a graph G and an edge e such that $\Omega(|V|^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used.

(c) Describe an algorithm for updating the transitive closure as edges are inserted into the graph. For any sequence of x insertions, your algorithm should run in total time $\sum_{i=1}^{x} t_i = O(|V|^3)$, where t_i is the time to update the transitive closure upon inserting the ith edge. Prove that your algorithm attains this time bound.

Problem 4

(a) Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

(b) Draw the recursion tree for the MergeSort algorithm on an array of 16 elements. Explain why memoization fails to speed up a good divide-and-conquer algorithm such as MergeSort.

(c) Imagine that you wish to exchange one currency for another. You realize that instead of directly exchanging one currency for another, you might be better off making a series of trades through other currencies, winding up with the currency you want. Suppose that you can trade n different currencies, numbered $1 \cdots n$, where you start with currency 1 and wish to wind up with currency n. You are given, for each pair of currencies i and j, an exchange rate r_{ij}, meaning that if you start with d units of currency i, you can trade for dr_{ij} units of currency j. A sequence of trades may entail a commission, which depends on the number of trades you make. Let c_k be the commission that you are charged when you make k trades. Show that if commissions c_k are arbitrary values, then the problem of finding the best sequence of exchanges from currency 1 to currency n does not necessarily exhibit optimal substructure.

Problem 5

Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., for example, $n = 6$ and the values are 1, 5, 10, 25, 50 and 100 cents.) Your beloved and benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.

(a) In the United States, there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin and recursively give change for the remainder. El Generalissimo does not approve of American capitalist greed. Show that there is a set of coin values for which the greedy algorithm does *not* always give the smallest possible of coins.

(b) Suppose El Generalissimo decides to impose a currency system where the coin denominations are consecutive powers $b^0, b^1, b^2, \cdots, b^k$ of some integer $b \geq 2$. Prove that despite El Generalissimo's disapproval, the greedy algorithm described in part (a) does make optimal change in this currency system.

(c) Design an algorithm to determine, given a target amount T and a sorted array $c[1 \cdots n]$ of coin denominations, the smallest number of coins needed to make T cents in change. Assume that $c[1] = 1$, so that it is possible to make change for any amount T. Remember to analyze the time complexity of your proposed algorithm. For full credit, your algorithm should have $O(nT)$ runtime.

Problem 6

A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or RACECAR, or AMANAPLANACATACANALPANAMA.

(a) Describe and analyze an algorithm to find the length of the *longest subsequence* of a given string that is also a palindrome. For example, the longest palindrome subsequence of the string

MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM

is MHYMRORMYHM; thus, given that string as input, your algorithm should return 11. For full credit, your algorithm should have $O(n^2)$ runtime for an input string of length n.

(b) Describe and analyze an algorithm to find the length of the *shortest supersequence* of a given string that is also a palindrome. For example, the shortest palindrome supersequence of TWENTYONE is TWENTOYOTNEWT, so given the string TWENTYONE as input, your algorithm should return 13. For full credit, your algorithm should have $O(n^2)$ runtime for an input string of length n.

Problem 7

This problem asks you to devise algorithms to compute optimal subtrees in *unrooted* trees—connected acyclic undirected graphs. In this problem. a *subtree* of an unrooted tree is any connected subgraph.

(a) Suppose you are given an unrooted tree T with weights on its *edges*, which may be positive, negative, or zero. Describe and analyze an algorithm to find a *path* in T with maximum total weight. Your algorithm only need to return the weight value. You may assume a path only contains distinct vertices. For full credit, your algorithm should have $O(n)$ runtime assuming T contains n vertices.

(b) Suppose you are given an unrooted tree T with weights on its *vertices*, which may be positive, negative, or zero. Describe and analyze an algorithm to find a subtree of T with maximum total weight. Your algorithm only need to return the weight value. For full credit, your algorithm should have $O(n)$ runtime assuming T contains n vertices. (This was a 2016 Google interview question.)

(We only ask for returning the weight value so as to keep the code cleaner. Nonetheless, you should also think about how to efficiently reconstruct the optimal solution. It's another good exercise!)