EE360C: Algorithms
University of Texas at Austin — Homework #5
Dr. David Soloveichik — Due: October 25, 2018 (*in-class quiz*)

Homework #5

You do not need to turn in these problems. The goal is to reinforce what we learned in class, as well as to cover material we didn't have time to cover in class. The in-class quiz that will cover the same or similar problems. Material on homework can also appear on exams.

Problem 1: Bottleneck Edges in Minimum Spanning Trees

One of the basic motivations behind the Minimum Spanning Tree Problem is the goal of designing a spanning network for a set of nodes with minimum *total* cost. Here we explore another type of objective: designing a spanning network for which the *most expensive* edge is as cheap as possible.

Specifically, let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of G; we define the *bottleneck edge* of T to be the edge of T with the greatest cost.

A spanning tree T of G is a *minimum-bottleneck spanning tree* if there is no spanning tree T' of G with a cheaper bottleneck edge.

(a) Is every minimum bottleneck tree of G a minimum spanning tree of G? Prove or give a counter example.

Solution

This is false. Let G have vertices $\{v_1, v_2, v_3\}$, with three edges (triangle). Suppose the edges $\{v_1, v_2\}$ and $\{v_1, v_3\}$ have weight 2, while edge $\{v_2, v_3\}$ has weight 1. Then every spanning tree has a bottleneck edge of weight 2 (to reach v_1), so the tree consisting of the two weight 2 nodes is a minimum bottleneck tree. It is not a minimum spanning tree, however, since its total weight is greater than that of the tree with one weight 2 edge and one weight 1 edge.

(b) Is every minimum spanning tree of G a minimum bottleneck tree of G? Prove or give a counter example.

Solution

This is true. Suppose that T is a minimum spanning tree of G and T' is a spanning tree with a lighter bottleneck edge. Thus T contains an edge e that is heavier than every edge in T'. So if we add e to T', it forms a cycle C on which it is the heaviest edge (since all other edges in C belong to T'. By the cycle property, then e does not belong to any minimum spanning tree, contradicting the fact that it is in T and T is a minimum spanning tree.

Problem 2: Coin Changing

Consider the problem of making change for n cents using the *fewest* number of coins.

(a) When people give change, they usually use the following greedy strategy: Select the largest coin denomination that is less than n (say it is c), and use one of these coins. Repeat for $n - c$ cents. Prove that this greedy algorithm gives the optimal solution for US coins: quarters,

dimes, nickels, and pennies.
Hint: Prove that an optimal solution for $n \geq 25$ must include at least 1 quarter, that an optimal solution for $10 \leq n < 25$ must include at least one dime, and that an optimal solution for $5 \leq n < 10$ must include at least one nickel.

Solution

At a step of the algorithm, we say a particular choice of coin is safe if it preserves the invariant that the coins chosen so far form a subset of an optimal solution. We show that every choice the greedy algorithm makes is safe, which will then guarantee that the final set of coins will be an optimal solution. (Note the similarity to the proof strategy for Kruskal's/Prim's MST algorithms where we were picking edges that we know must be part of the optimal solution.)
We will argue that an optimal solution for $n \geq 25$ must include at least 1 quarter. Thus the greedy choice in this case is safe. Similarly, we'll argue that an optimal solution for $10 \leq n < 25$ must include at least one dime, and that an optimal solution for $5 \leq n < 10$ must include at least one nickel, implying that the greedy choice in every case is safe.
Note the following properties of optimal solutions: No optimal solution can include 5 pennies. Similarly, no optimal solution can include 2 nickels, or 1 nickel and 2 dimes. (These can be replaced with 1 nickel, 1 dime, and 1 quarter, respectively.) Finally, we note that no optimal solution can include 3 dimes because this can be replaced with 1 quarter and 1 nickel. This implies that the largest optimal solution not including quarters is for $n = 24$ (2 dimes + 4 pennies). Thus an optimal solution for $n \geq 25$ must include at least 1 quarter. Similarly, the largest optimal solution not including quarters or dimes is for $n = 9$ (1 nickel + 4 pennies). Thus an optimal solution for $10 \leq n < 25$ must include at least one dime. Finally, the largest optimal solution not including quarters, dimes, or nickels is for $n = 4$ (4 pennies). Thus an optimal solution for $5 \leq n < 10$ must include at least one nickel.

(b) Although this greedy algorithm is always optimal for US coins, it may not be for other sets of coin denominations. Give a set of coin denominations for which your greedy algorithm does not yield an optimal solution. Your set should include a penny to ensure that you can always successfully make change. (If you are establishing a new currency, you probably want to avoid such sets of coin denominations!)

Solution

One example is a scheme with a 1 cent coin, a 6 cent coin, and an 8 cent coin. To make 12 cents, the greedy algorithm would use 5 coins, while the optimal algorithm would use 2. Another example is if you remove the nickel from US currency and try to give change for 30.

Problem 3: Huffman Coding

In this problem, we are going to work toward proving the optimality of the Huffman Algorithm. (This is covered in Section 4.8 of the textbook.)

Solution

Please see the textbook Section 4.8 where these results are worked out.

(a) All the prefix trees we saw in class have the following property: a node is either a leaf or it has 2 children. Prove that a binary tree in which there is a node with only 1 child cannot correspond to an optimal prefix code.

(b) Prove that there is always an optimal prefix tree in which the two lowest frequency symbols are assigned to leaves with the same parent.

(c) Prove that the Huffman algorithm achieves the minimum average number of bits per letter of any prefix code.
In your proof you can use the following fact: Let T be a binary prefix tree. Suppose x and y are two leaves with the same parent. Let T' be the tree formed from T by replacing x, y and their parent with a leaf z, and making $f(z) = f(x) + f(y)$. Then $ABL(T) = ABL(T') + f(z)$.
Hint: Part (b) is also used in the proof.