CS 577: Introduction to Algorithms — Homework 5

Out: 10/15/10 — **Due: 10/22/10**

Ground Rules

- **Grading.** You will be graded on the correctness as well as clarity of your solutions. You are required to prove any claims that you make. In particular, when you are asked to design an algorithm, you must argue its correctness as well as running time. You may use without proof any theorems proven in class or in the textbook, as long as you state and cite them properly.
- **Collaboration.** You may work on and submit solutions in pairs.
- **Lateness.** Homework is due promptly at the start of class. Late homework will receive zero credit.
- **Extra credit questions.** Extra credit questions will not directly contribute to your score. However, they will be taken into account in the final grading and may improve your overall grade if your total score is close to the boundary between two grades. Furthermore they are fun to solve and improve your understanding of the course.
- Start working on your homework early. Plan your work in such a way that you have the opportunity to put some problems on the back burner for a while and revisit them later. Good luck!

Problems

1. **(3+2 points)** One of your friends who has not taken an algorithms class is working at a cafe downtown and is often confronted with the problem of making change. Your friend would like a simple proceedure for doing this that uses the minimum number of coins.

   (a) Give a greedy algorithm that is optimal when the available coins are pennies, nickles, dimes, and quarters. Prove that your algorithm is optimal.

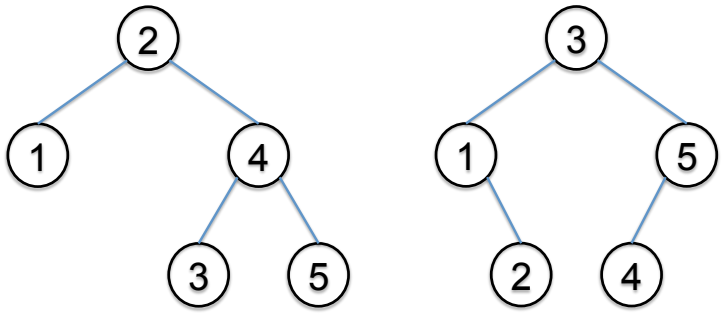
   (b) Another friend overheared you decribing your algorithm (and because it was so simple) she remembered it when on her study abroad program on Mars. When she returns she complains that she tried to use the algorithm but it did not work. Mars, remember, has a different currency that uses different denominations than the United States. Show that this is possible by giving a set of denominations that breaks the greedy algorithm.

2. **(5 points)** In class, we developed an $O(nC)$ time algorithm for the Knapsack problem, where n is the number of items and C is the capacity of the knapsack. Give an algorithm for the Knapsack problem that runs in time $O(nV)$ where V is the total value of all the items (i.e. $V = \sum_{i \leq n} v_i$). The running time of your algorithm should be independent of C.

3. **(5 points)** One way to search efficiently is to use binary search trees. A binary search tree is a binary tree where each node is labeled with one of the elements and the labels satisfy the following property — the label of every node is larger than the label of every node in its left subtree, and smaller than the label of every node in its right subtree. Two examples of binary search trees over the elements $\{1, 2, 3, 4, 5\}$ are given below.

   Given the frequencies $f_1, \cdots, f_n$ of searching for elements $1, \cdots, n$, the average cost of search is equal to $\sum_i f_i \ell_i$ where ℓ_i is the "depth" of element i, or the length of the path from the root to the node labeled i. Binary search trees are therefore just like Huffman codes with two differences—elements can be assigned to internal nodes, and the assignment must satisfy the ordering described above.

   Give an algorithm for constructing the optimal binary search tree (minimizing the average cost of search) given n elements $1, \cdots, n$ and their frequencies $f_1, \cdots, f_n$. Your algorithm should run in time polynomial in n.



4. **(Extra credit.)** Give an $O(n^2)$ time algorithm for the optimal binary tree problem using the following fact (that you do not need to prove): Let $r(i, j)$ denote the index of the root of the optimal tree over elements $i, i+1, \cdots, j$; Then $r(i, j-1) \leq r(i, j) \leq r(i+1, j)$ for all $i < j$.