CS 577: Introduction to Algorithms — Homework 4

Out: 02/17/17 — **Due: 02/24/17**

Ground Rules

- Review problems will be discussed at the following week's review.
- Self-graded problems will be discussed at the following week's review, and are to be self-graded by you during the review. Only students present at the review session and turning in their graded solutions in person will earn points.
- Graded problems are to be turned in on the due date at the beginning of the lecture.
- Self-graded problems should be done individually.
- Graded problems should be done in pairs.
- Please follow page limits for all self-graded and graded problems.
- Write your name(s) clearly on your submissions, and turn in each problem on **separate sheets of paper**.

Review Problems

1. Let G be a connected graph, and let T be a depth-first spanning tree of G rooted at some node v. Prove that if T is also a breadth-first spanning tree of G rooted at v, then $G = T$.
2. (Taken from "Algorithms, Etc.") Whenever groups of pigeons gather, they instinctively establish a pecking order. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles–for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.
   (a) Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. We call such an arrangement a pecking order. (Hint: use induction.)
   (b) It is possible to convert your inductive proof from part (a) into an efficient recursive algorithm that will actually find a pecking order over pigeons. We will take a graph-theoretic approach, however.
   Suppose you are given a directed graph representing the pecking relationships among a set of n pigeons. The graph contains one vertex per pigeon, and it contains an edge $i \to j$ if and only if pigeon i pecks pigeon j. Prove that if you run DFS over the outgoing edges starting from an appropriate vertex, and output nodes in the order in which they were finished by the algorithm, then you get a pecking order in reverse (i.e. right to left).

Self-graded Problems

3. **(Page limit: 1 sheet; 2 sides)** For the following two problems, give a recursive equation (a.k.a. principle of optimality) for the function to be computed. No proofs or algorithms or run time analyses are required.
   (a) Martian currency has n different coins with integral denominations $a_1, \cdots, a_n$. For a given integer value A, your goal is to make change for a total amount of A using the fewest number of coins possible. Your function should return the fewest number of coins needed to make change.
   (b) You are given an arithmetic expression containing n integers and $n-1$ operators, each either $+$, $-$, or $\times$. Your goal is to perform the operations in an order that maximizes the value of the expression. Your function should output this maximum value.
   For example:

- For the expression $6 \times 3 + 2 \times 5$, the optimal ordering is to add the middle numbers first, then perform the multiplications: $((6 \times (3 + 2)) \times 5) = 150$.
- For the expression $(-3) \times 3 + 3$, the optimal ordering is $(((-3) \times 3) + 3) = -6$.
- For the expression $(-3) \times 3 - 3$, the optimal ordering is $((-3) \times (3 - 3)) = 0$.

Graded Problems

4. **(Page limit: 2 sheets; 3 sides)** Consider the following card game between two players. The dealer deals n cards face up in a row. Each card has a value visible to both players. The players move in sequence. Each player, at her turn, can pick either the rightmost or the leftmost of the remaining cards on the table. The game ends when no cards are left. The goal of each player is to accumulate as much value as possible.

   For example, suppose that $n = 5$ and the cards have values $3, 2, 1, 7, 4$ in order from left to right. Then suppose the players pick the cards in the order $\mathbf{4}, 7, \mathbf{3}, 2, \mathbf{1}$, where boldface indicates the first player's pick, and the remainder the second player's pick. Then the first player gets total value $4 + 3 + 1 = 8$ and the second gets value $7 + 2 = 9$. On the other hand, if the players pick the cards in the order $\mathbf{3}, 4, \mathbf{7}, 2, \mathbf{1}$, then the first player gets total value $3 + 7 + 1 = 11$ and the second gets value $4 + 2 = 6$. So, the first player should start by picking the leftmost card with value 3.

   Design an $O(n^2)$ time algorithm for computing the optimal strategy for the first player. In particular, given a list of values, your algorithm should output whether the first player should pick the leftmost or the rightmost card in order to maximize her total value. Assume that the second player also plays optimally.

   Give a brief argument for the correctness of your recursive equation as well as a brief argument bounding the running time of your algorithm.

5. **(Page limit: 2 sheets; 3 sides)** (Taken from "Algorithm Design") Consider the following inventory problem. You are running a company that sells some large product (say trucks), and predictions tell you the quantity of sales to expect over the next n months. Let d_i denote the number of sales you expect in month i. We'll assume that all sales happen at the beginning of the month, and trucks that are not sold are *stored* until the beginning of the next month. You can store at most S trucks, and it costs C to store a single truck for a month. You receive shipments of trucks by placing orders for them, and there is a fixed ordering fee of K each time you place an order (regardless of the number of trucks you order). You start out with no trucks. The problem is to design an algorithm that decides how to place orders so that you satisfy all the demands $\{d_i\}$, and minimize the costs.

   To summarize:

   - Your algorithm receives as input the demands $\{d_i\}$, the costs C and K, and the inventory limit S.
   - Your algorithm should output how many trucks to order in each of the n months; you may skip ordering in some of the months.
   - In each month you need enough trucks to satisfy the demand d_i but the number left over after satisfying the demand for the month should not exceed the limit S.
   - Your total cost is K times the number of orders placed plus the sum over the months of C times the number of trucks left over after sales that month.

   Give a DP-based algorithm that solves this problem in time polynomial in n. (Solutions with a running time that depends on parameters other than n will receive partial credit.)

   Give a brief argument for the correctness of your recursive equation as well as a brief argument bounding the running time of your algorithm.