Homework 4

2019.03.21

Note: When the exercise asks you to "design an algorithm for…," it always means that "designs an EFFICIENT algorithm for … and ANALYZES your algorithm". You should keep this in mind when writing solutions.

1. **Exercises 15.1-2**

   Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the ***density*** of a rod of length i to be p_i / i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \le i \le n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n-i.

2. **Exercises 15.1-3**

   Consider a modification of the rod-cutting problem in which, in addition to a price pi for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

3. **Exercises 15.1-4**

   Modify MEMORIZED-CUT-ROD to return not only the value but the actual solution, too.

4. **Exercises 15.2-1**

   Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is《5, 10, 3, 12, 5, 50, 6》.

5. A mathematical expression is given without parentheses. Design an algorithm to parenthesize the expression such that the value of the expression is maximized. For example, consider the expression: 2+7×5. There are two ways to parenthesize the expression 2+(7×5) = 37 and (2+7)×5 = 45, so in this case, your algorithm should output the second expression. Here, you may assume the given expressions contain only 3 kinds of binary operators '+', '–', and '×'.

6. **Rod Cutting Problem**

   Suppose you have a rod of length M, and you want to cut up the rod and sell the pieces in a way. A piece of length i is worth p_i dollars, when $i \le n$. Now M is greater than n. Assume the length of a rod is always an integer. Design an algorithm to maximize the total profit you get.

7. **Exercises 15.3-1**
Which is a more efficient way to determine the optimal number of multiplications in a matrix-chain multiplication problem: enumerating all the ways of parenthesizing the product and computing the number of multiplications for each, or running RECURSIVE-MATRIX-CHAIN? Justify your answer.

```
RECURSIVE-MATRIX-CHAIN (p, i, j)
    return 0 if i == j
    m[i][j] = ∞
    for k in i...j
        q = RECURSIVE-MATRIX-CHAIN (p, i, k) +
            RECURSIVE-MATRIX-CHAIN (p, k + 1, j) +
            p[i - 1] * p[k] * p[j]
        m[i][j] = q if q < m[i][j]
    return m[i][j]
```

8. **Exercises 15.3-4**
As stated, in dynamic programming we first solve the sub-problems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that it is not always necessary to solve all the sub-problems in order to find an optimal solution. She suggests that an optimal solution to the matrix-chain multiplication problem can be found by always choosing the matrix A_k at which to split the sub-product $A_i A_{i+1} \dots A_j$ (by selecting k to minimize the quantity $p_{i-1} p_k p_j$) before solving the sub-problems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a sub-optimal solution.