Algorithm HW4

1. 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 $1 \leq i \leq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n-i.

2. Consider a modification of the rod-cutting problem in which, in addition to a price p_i 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. Modify MEMORIZED-CUT-ROD to return not only the value but the actual solution, too.

4. 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. 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]
```

7. 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 $A_iA_{i+1}...A_j$ (by selecting k to minimize the quantity $p_{i-1}p_kp_j$) *before* solving the sub-problems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a sub-optimal solution.