Homework 4
CSC 311
Spring 2020/1441-1442H

March 12, 2020

Problem 1

Given matrices

$$A_1, A_2, \cdots, A_n$$

Of dimensions $p_1 \times p_2, p_2 \times p_3, \cdots, p_n \times p_{n+1}$, your friend Ahmed says that he has two simpler ways to solve the Matrix Chain Multiplication problem, instead of dynamic programming. For each of the two given strategies, *give a counter example showing that it is sub-optimal.*

1. At each step, perform the matrix multiplication which costs the least number of scalar multiplications at this step.

2. Consider the product ABC of matrices of dimensions 2×100, 100×3 and 3×4. Ahmed noticed that if you multiply it as $(AB)C$, you perform fewer scalar multiplications than $A(BC)$ because

$$2 \times 100 \times 3 + 2 \times 3 \times 4 < 100 \times 3 \times 4 + 2 \times 100 \times 4 \qquad (1)$$

   So, he suggests that we always find the maximum p_i and perform the multiplication around it. This way, 100 only shows up in *one* term (compare the two sides of inequality 1). In this case, the maximum was $p_2 = 100$ so we perform the multiplication (AB) first.

Problem 2: Change making

Given an amount n, change n into the *least* number of coins, where coins have denominations $d_1 < d_2 < d_3 < \cdots < d_m$. The solution to this problem will be in the form of $c_1, c_2, \ldots, c_m$, quantities for each coin denomination, while minimizing the *total number* of coins, $F(n) = \sum_{i=1}^{m} c_i$.

1. For the coin denominations $1, 5, 10, 25, 50$, give a simple algorithm (without dynamic programming) to output the number of coins for each denomination, minimizing the total number of coins.

2. For the coin denominations $1, 4, 10, 22, 40$, give an example to show that the simple algorithm does not work for all coin denomination assignments.

3. For a general denomination assignment $d_1 < d_2 < \cdots < d_m$, where $d_1 = 1$, define the *value* $F(n)$ of an optimal solution recursively.

4. Show the optimal substructure.

5. Give a recursive definition of $F(n)$ and a top-down, memoization algorithm using dynamic programming to calculate $F(n)$.

6. Modify your algorithm to also produce the quantities of each coin denomination $c_1, c_2, \ldots, c_m$.

Problem 3 : Well placement

You are tasked with choosing how to dig wells for n farms arranged in a strip. Each farm has productivity p_i, indicating how many potatoes it would produce if it had a well. However, if you place wells in any two adjacent farms, you risk overusing and drying up the underground water deposits before potatoes can grow. Therefore, you can only place wells in *non-adjacent* farms.

Given the productivity $p_1, p_2, \ldots, p_n$ of each of the n farms, where do we place the wells?

1. Does this problem have optimal substructure? Define the structure of an optimal solution to show the optimal substructure.

2. Prove the optimal substructure.

3. Give a recursive definition for the *value* of an optimal solution

4. Give a recursive algorithm (without memoization) to solve the problem.

5. Give a bottom-up, dynamic programming algorithm to solve this problem.

6. What is the time complexity of your bottom-up algorithm?

Problem 4: Weighted job scheduling

You are given a set of n jobs. Each job i has a beginning time , s_i, and a finishing time f_i. Each job also has a weight w_i. Your task is to design an algorithm to choose a set of jobs which are *mutually independent*, and with maximum weight. Two jobs are independent if they are not active at the same time. Suppose that your jobs are unique, and sorted in increasing order of f_i. That is, $f_1 < f_2 < f_3 < \ldots < f_n$ (Hint: For each job, define $\rho(i)$ to be *the job with the highest index, which is independent with job* i)

1. Can you find the optimal substructure in this problem?
2. Give a recursive definition for the value of the optimal solution.
3. Give either a bottom up, or top-down algorithm to solve this problem.