**CS3230 Design and Analysis of Algorithms** August 22, 2020

# Problem Set 2

*Lecturer: Divesh Aggarwal*

## Instruction

This tutorial contains two kinds of questions - questions marked with an asterisk (*) are part of the assignment, and you need to submit your solutions to these questions; solutions to unmarked questions need not be submitted, but you should still attempt them. Your assignment solution will be graded. Submit your assignment solution by uploading to the LumiNUS. Solutions typeset using LaTeXare encouraged but not necessary. You may also submit scanned handwritten solutions, but they must be clearly readable. Solutions that are not readable may receive only partial marks. Late submissions will receive no marks. This assignment solution will be due by 11:59 pm, September 5, 2020 (Saturday).

---

## Problem 2-1 (Dollar-Notes)

In a previous life, you worked as a cashier in the lost Antarctican colony of Nadiria, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource in Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadiria, called Dream-Dollars, was available in the following denominations: $1, $4, $7, $13, $28, $52, $91, and $365.

(a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make $122 using the greedy algorithm, we first take a $91 bill, then a $28 bill, and finally three $1 bills. Give an example where this greedy algorithm uses more Dream-Dollar bills than the minimum possible. [Hint: It may be easier to write a small program than to work this out by hand.]

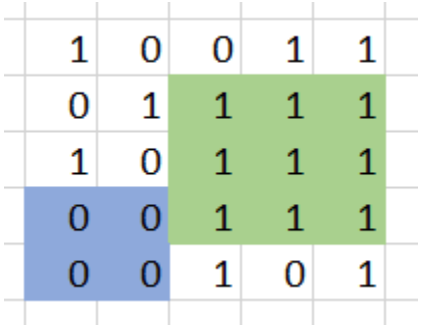
(b) Describe and analyze a recursive algorithm that computes, given an integer k, the minimum number of bills needed to make k Dream-Dollars. (Don't worry about making your algorithm fast; just make sure it's correct.)

(c) Describe a dynamic programming algorithm that computes, given an integer k, the minimum number of bills needed to make k Dream-Dollars. (This one needs to be fast.)

## Problem 2-2 (*Square Solid Block) 10 Points

Given a 2-dimensional array $M[1 \ldots n][1 \ldots n]$ consisting of 0s and 1s, a *square solid block* in $M$ is a subarray in the form $M[(i-\ell+1) \ldots i][(j-\ell+1) \ldots j]$ in which all bits are equal (all 1 or all 0). In other words, $\ell$ is the length of the square block.

Figure 1: An example where $n = 5$.




The given diagram is an example where $n = 5$. The green block $M[2 \ldots 4][3 \ldots 5]$ is a *square solid block* of length 3 and it is the largest *square solid block*. The blue block $M[4 \ldots 5][1 \ldots 2]$ is a *square solid block* of length 2. Each individual cell is a *square solid block* of length 1.

For $1 \leq i, j \leq n$, let $f[i, j]$ be the length $\ell$ of the **largest square solid block** whose right bottom cell is $M[i][j]$, i.e., the largest $\ell$ such that $M[(i-\ell+1) \ldots i][(j-\ell+1) \ldots j]$ is a solid square block.

(a) (5 Points) By filling in the blanks (or otherwise), give a recursive formula for $f[i, j]$. Remember to define the base case. Prove that your recursive solution is correct. If you have a correct recursive solution, you should be able to prove it in just 2-3 sentences.

$$f[i,j] := \begin{cases} 1 & \text{if } \cdots \\ 1 + \min(\ldots, \ldots, \ldots) & \text{otherwise} \end{cases}.$$

(b) (5 Points) Based on your recursive formula, **write pseudocode** for an $O(n^2)$ time bottom-up dynamic programming algorithm to find the maximum length of a *square solid block* in $M$. For example, in the example in Figure 1, your algorithm should output 3. Partial credit is given for algorithms with a worse running time.

## Problem 2-3 (SubsetSum)

Describe recursive algorithms for the following generalizations of the SubsetSum problem

(a) Given an array $X[1..n]$ of positive integers and an integer $T$, compute the *number* of subsets of $X$ whose elements sum to $T$.

(b) Given two arrays $X[1..n]$ and $W[1..n]$ of positive integers and an integer $T$, where each $W[i]$ denotes the weight of the corresponding element $X[i]$, compute the maximum weight subset of $X$ whose elements sum to $T$ and output its weight. If no subset of $X$ sums to $T$, your algorithm should return $-\infty$.

## Problem 2-4 (*Multiplying to Hit a Given Target) 15 Points

Let $\star : \{1, \ldots, k\} \times \{1, \ldots, k\} \mapsto \{1, \ldots, k\}$ be a binary operation. Below we assume the values of $a \star b$ for $a, b \in \{1, \ldots, k\}$ are stored in some $k \times k$ array $M$ such that $M[a][b] = a \star b$. Consider the

problem of examining a string $x = x_1 x_2 \ldots x_n$, where each $x_i \in \{1, \ldots, k\}$, and deciding whether or not it is possible to parenthesize the expression $x_1 \star x_2 \star \ldots \star x_n$ in such a way that the value of the resulting expression is a given target element $t \in \{1, \ldots, k\}$. Notice, the multiplication table is neither commutative or associative, so the order of multiplication matters (and, hence, the result of the expression is not even well defined unless a complete "parenthesization" is specified). For example, consider the following multiplication table and the string $x = 2221$.

Table 1: Multiplication table

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 1 | 3 | 3 |
| 2 | 1 | 1 | 2 |
| 3 | 3 | 3 | 3 |

Parenthesizing it $(2 \star 2) \star (2 \star 1)$ gives $t = 1$, but $((2 \star 2) \star 2) \star 1$ gives $t = 3$. On the other hand, no possible parenthesization gives $t = 2$ (you may check this).

(a) (10 Points) Assume you are given as input the following: $n$, $k$, $t$, $x[1 \ldots n]$ and $M$. Give a dynamic programming algorithm that runs in time $O(n^3 k^2)$ and outputs YES if there exists a paranthesization for $x$ that results in the product equal to $t$, and NO otherwise. For instance, in the above example with $x = 2221$, the answer is YES if $t = 1$ or $t = 3$, but NO if $t = 2$.

(b) (5 Points) Analyze the running time of your algorithm.

## Problem 2-5 (Bracketings)

Imagine a unary alphabet with a single letter $x$. A (valid) *bracketing* $B$ is a string over three symbols $x, (, )$ defined recursively as follows: (1) a single letter $x$ is a bracketing, and (2) for any $k \geq 2$, if $B_1, \ldots, B_k$ are (valid) bracketings, then so is $B = (B_1 B_2 \ldots B_k)$. A bracketing $B$ is called *binary* if rule (2) can only be applied with $k = 2$. Then length $n$ of $B$ is the number of $x$'s it has (i.e., one ignores the parenthesis).

For example, there are 11 possible bracketings of length $n = 4$: $(xxxx)$, $((xx)xx)$, $((xxx)x)$, $(x(xxx))$, $(x(xx)x)$, $(xx(xx))$, $((xx)(xx))$, $(x(x(xx)))$, $((x(xx))x)$, $(x((xx)x))$, $(((xx)x)x)$, of which *only the last five* are binary.

(a) Let $b(n)$ denote the number of binary bracketings of length $n$. Show that $b(n)$ is given by the following recurrence:

$$b(n) = \sum_{i=1}^{n-1} b(i) b(n-i) \ .$$

(b) Use the result from part (a) to give an $O(n^2)$-time algorithm to compute $b(n)$ given $n$ as input. Assume that multiplication of two integers takes time $O(1)$.

(c) Generalize part (a) and (b) by giving a similar recurrence(with proof) as part (a) to find the total number $f(n)$ of bracketings of length $n$, and then give an $O(n^2)$-time algorithm to compute $f(n)$.