CS 170 HW 7

Due on 2019-03-11, at 10:00 pm

1 Study Group

List the names and SIDs of the members in your study group.

2 Steel Beams

You're a construction engineer tasked with building a new transit center for a large city. The design for the center calls for a T-foot-long steel beam for integer $T > 0$. Your supplier can provide you with an *unlimited* number of steel beams of integer lengths $0 < c_1 < \ldots < c_k$ feet. You can weld as many beams as you like together; if you weld together an a-foot beam and a b-foot beam you'll have an $(a + b)$-foot beam. Unfortunately, every weld increases the chance that the beam might break, so you want as few as possible.

Your task is to design an algorithm which outputs how many beams of each length you need to obtain a T-foot beam with the minimum number of welds, or 'not possible' if there's no way to make a T-foot beam from the lengths you're given. (If there are multiple optimal solutions, your algorithm may return any of them.)

(a) Consider the following greedy strategy. Start with zero beams of each type. While the total length of all the beams you have is less than T, add the longest beam you can without the total length going over T.

(i) Suppose that we have 1-foot, 2-foot and 5-foot beams. Show that the greedy strategy always finds the optimum.

(ii) Find a (short) list of beam sizes $c_1, \ldots, c_k$ and target T such that the greedy strategy fails to find the optimum. Briefly justify your choice.

(b) Give a dynamic programming algorithm which always finds the optimum. Describe your algorithm, including a clear statement of your recurrence, show that it is correct and prove its running time. How much space does your algorithm use?

3 Egg Drop

You are given two identical eggs and a hundred story building. You need to figure out the maximum floor $\ell \in \{1, \ldots, 100\}$ that you can drop them from without breaking them. Each egg will break if dropped from a floor greater than or equal to ℓ, will never break when dropped from a floor less then ℓ, and once an egg breaks, you cannot use it any more.

(a) What is a strategy that takes the fewest number of drops to figure out what ℓ is, no matter what value ℓ is? What is the maximum number of drops in this strategy?

(b) What if you had an n story building? What if you had k eggs?

4 Non-Prefix Code

As we have learned in lecture, the Huffman code satisfies the *Prefix Property*, which states that the bit string representing each symbol is not a prefix of the bit string representing any other symbol. One nice property of such codes is that, given a bit string, there is at most one way to decode it back to a sequence of symbols. However, this is not true anymore once we are working with codes that do not satisfy the Prefix Property. For example, consider the code that maps A to 1, B to 01 and C to 101. A bit string 101 can be interpreted in two ways: as C or as AB.

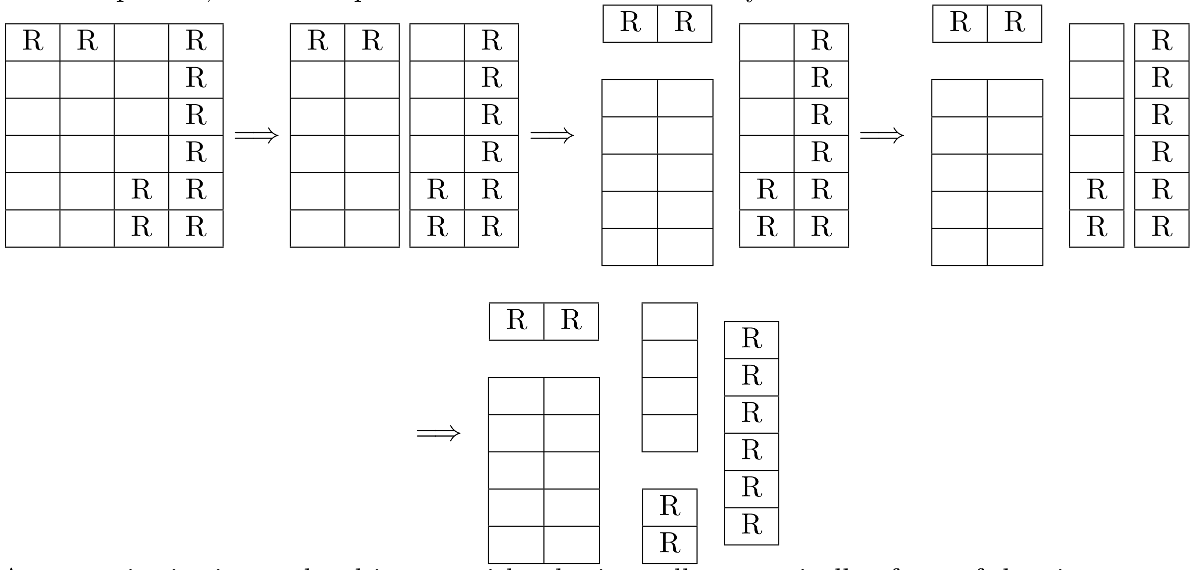
Your task is to, given a bit string s, determine how many ways one can interpret s. The mapping from symbols to bit strings of the code will be given to you as a dictionary d (e.g., in the example, $d = \{A : 1, B : 01, C : 101\}$); you may assume that you can access each symbol in the dictionary in constant time. Your algorithm should run in time at most $O(nm\ell)$ where n is the length of the input bit string s, m is the number of symbols, and ℓ is an upper bound on the length of the bit strings representing symbols.

Clearly describe your algorithm, prove its correctness and runtime.

5 Breaking Chocolate

There is a chocolate bar consisting of an $m \times n$ rectangular grid of squares. Some of the squares have raisins in them, and you hate raisins. You would like to *break* the chocolate bar into pieces so as to separate all the squares with raisins, from all the squares with no raisins.

For example, shown below is a 6×4 chocolate bar with raisins in squares marked R. As shown in the picture, one can separate the raisins out in exactly four *breaks*.



(At any point in time, a *break* is a cut either horizontally or vertically of one of the pieces at the time)

Design a DP based algorithm to find the smallest number of breaks needed to separate all the raisins out. Formally, the problem is as follows:

Input: Dimensions of the chocolate bar $m \times n$ and an array $A[i,j]$ such that $A[i,j] = 1$ if and only if the ij^{th} square has a raisin.

Goal: Find the minimum number of breaks needed to separate the raisins out.

(a) Define your subproblem.

(b) Write down the recurrence relation for your subproblems.

(c) What is the time complexity of solving the above mentioned recurrence? Provide a justification for the same.

6 Road Trip

Suppose you want to drive from San Francisco to New York City on I-80. Your car holds C gallons of gas and gets m miles to the gallon. You are handed a list of the n gas stations that are on I-80 and the price that they sell gas. Let d_i be the distance of the i^{th} gas station from SF, and let c_i be the cost of gasoline at the i^{th} station. Furthermore, you can assume that for any two stations i and j, the distance $|d_i - d_j|$ between them is divisible by m. You start out with an empty tank at station 1. Your final destination is gas station n. You may not run out of gas between stations but you need not fill up when you stop at a station, for example, you might to decide to purchase only 1 gallon at a given station.

Find a polynomial-time dynamic programming algorithm to output the minimum gas bill to cross the country. Clearly describe your algorithm and prove its correctness. Analyze the running time of your algorithm in terms of n and C.

7 Propositional Parentheses

You are given a propositional logic formula using only $\wedge$, $\vee$, T, and F that does not have parentheses. You want to find out how many different ways there are to *correctly parenthesize* the formula so that the resulting formula evaluates to true.

A formula A is correctly parenthesized if $A = T$, $A = F$, or $A = (B \wedge C)$ or $A = (B \vee C)$ where B, C are correctly parenthesized formulas. For example, the formula $T \vee F \vee T \wedge F$ can be correctly parenthesized in 5 ways:

$$(T \vee (F \vee (T \wedge F))) \quad (T \vee ((F \vee T) \wedge F)) \quad ((T \vee F) \vee (T \wedge F))$$
$$(((T \vee F) \vee T) \wedge F) \quad ((T \vee (F \vee T)) \wedge F)$$

of which 3 evaluate to true: $((T \vee F) \vee (T \wedge F))$, $(T \vee ((F \vee T) \wedge F))$, and $(T \vee (F \vee (T \wedge F)))$.

(a) Give a dynamic programming algorithm to solve this problem. Describe your algorithm, including a clear statement of your recurrence, show that it is correct, and prove its running time.

(b) Briefly explain how you could use your algorithm to find the probability that, under a *uniformly randomly chosen* correct parenthesization, the formula evaluates to true.