CS 161: Homework 6

Due by November 19, 2016 at 11:59pm

Instructions: Please answer the following questions to the best of your ability. Provide full and rigorous proofs and include all relevant calculations. When writing proofs, please strive for clarity and brevity (in that order). Please see the course website for submission instructions and collaboration policy. Cite any sources that you use.

Remember that when you design an algorithm, in addition to an appropriate description of your algorithm, you must provide sufficient explanation for its correctness and analyze its running time. See the Homework Advice document on the course website for details.

In your submission, please start each question on a new page.

Question 1 (25 points)

Let a *randomly chosen binary search tree* on n keys be a binary search tree chosen uniformly at random from the set of all possible binary search trees of n keys. That is, each binary search tree of n keys is equally likely to be chosen. Without loss of generality, we assume the n keys are $\{1, \dots, n\}$.

(a) (5 points) Show that the notion of a randomly chosen binary search tree is different from the notion of a randomly built binary search tree given in Homework 4. (Hint: Consider the $n = 3$ case.)

(b) (6 points) Let $C(n)$ be the number of possible binary search trees on n keys. Find a recurrence for $C(n)$. Give base case(s).

(c) (7 points) Design a dynamic programming algorithm for computing $C(n)$. Briefly argue the correctness and runtime of your algorithm.

(d) (7 points) Design a recursive algorithm that returns a randomly chosen binary search tree on n keys, using the values $C(0), C(1), \dots, C(n)$ computed in part (c). Assume you have a function RANDOM that returns i with probability p_i given probabilities $(p_1, \dots, p_n)$ such that $\sum_i p_i = 1$. Briefly argue the correctness. No runtime analysis is required. (Hint: How do you choose the root?)

Question 2 (25 points)

In class, we learned how to find the longest common subsequence (LCS) of two strings. When computing the LCS, we were allowed to skip any number of characters in each of the input strings. In this question, we want to find the longest common contiguous subsequence (LCCS) of two strings. The LCCS of two strings is a maximal string that appears contiguously in both strings. For example, the LCCS with no gaps of 'GGCC' and 'GGAC' has length 2 ('GG'), while the LCS of these two strings has length 3 ('GGC').

We can solve this using dynamic programming. Assume the input is two strings, x and y, of lengths n and m, respectively. We define $A_0[i, j]$ to be be the length of the LCCS of x and y that its last character is aligned with $x[i]$ and $y[j]$. Formally, $A_0[i, j]$ is the length ℓ of the maximal string z such that for every $k = 0, \dots, \ell - 1$, $z[\ell - k] = x[i - k] = y[j - k]$.

In the example, x is 'GGCC' and y is 'GGAC'. A_0 contains the values:

$\mathbf{A_0}$	G	GG	GGA	GGAC
G	1	1	0	0
GG	1	2	0	0
GGC	0	0	0	1
GGCC	0	0	0	1

(a) (7 points) Write a recurrence for $A_0[i,j]$. What are the base cases? Prove the correctness of the recurrence by induction.

(b) (5 points) Design a dynamic programming algorithm that computes the length of the LCCS of two strings x, y.

You learn that in your input strings, a character was inserted by mistake, and you want to find the length of the LCCS that skips at most one character in one of the input strings. We define $A_1[i,j]$ to be be the length of the LCCS of x and y with at most one skip that its last character is aligned with $x[i]$ and $y[j]$. Formally, $A_1[i,j]$ contains the length ℓ of a maximal string z such that one of the following holds:

1. There are no skips: $z = x[i-\ell+1 : i] = y[j-\ell+1 : j]$.
2. There is one skip at $x[k]$: $z = x[i-\ell : k-1] + x[k+1 : i] = y[j-\ell+1 : j]$. Note that $i-\ell+1 \leq k \leq i$.
3. There is one skip at $y[k]$: $z = x[i-\ell+1 : i] = y[j-\ell : k-1] + y[k+1 : j]$. Note that $j-\ell+1 \leq k \leq j$.

In the example, A_1 contains the values:

$\mathbf{A_1}$	G	GG	GGA	GGAC
G	1	1	1	0
GG	1	2	2	0
GGC	1	2	0	3
GGCC	0	0	0	1

(c) (13 points) Design a dynamic programming algorithm that finds the length of the LCCS of two strings x, y with at most one skip. Prove correctness and analyze the running time.

Hint: Write a recurrence for $A_1[i,j]$ that can use values from A_0.

Question 3 (25 points)

You are a manager at an electronics manufacturing plant and are trying to think of ways to increase profits by automating the manufacturing process. The process of building the company's product is comprised of n independent stages (you can think of the product as being a combination of n different components), where each stage is performed by a robot R_i specialized for producing component i. Unfortunately, the robots can make mistakes and each have some probability f_i of failing to produce a component which meets the required standard (you may assume that failures are independent). To improve efficiency you decide to have redundancy by having m_i multiple copies of robot R_i at stage each i. Lastly, each robot R_i has a corresponding cost c_i and you have a total budget of B. Assume c_i and B are positive integers such that the budget allows to buy at least one robot for each stage ($B \geq \sum_{i=1}^{n} c_i$).

(a) (5 points) Given the probabilities $f_1, ..., f_n$ and the number of copies $m_1, ..., m_n$ what is the probability that the system successfully completes a product whose components meet all the standards?

(b) (10 points) You would like to come up with an efficient algorithm for determining the number of redundancies at each stage which are within the available budget and maximize the probability of completing a product. It turns out that you can think of the problem in terms of subproblems. Define a recurrence relation for solving an arbitrary subproblem as well as a base case.

(c) (10 points) Design a dynamic programming algorithm to solve the problem utilizing the recurrence you defined in part (b). Briefly argue correctness and analyze the run time of your algorithm.

Question 4 (25 points)

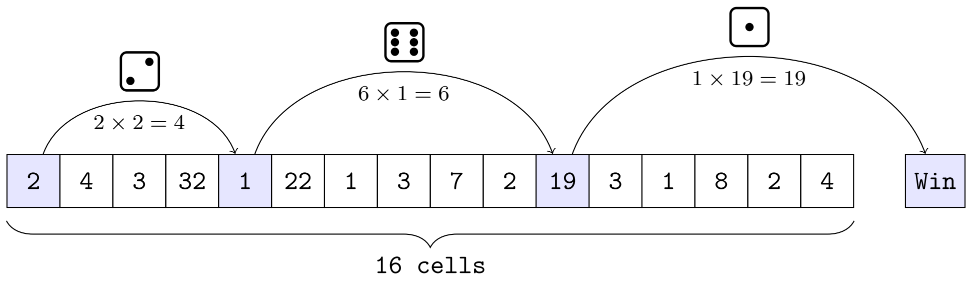
You are playing a boring and nondescript single-player board game. On the board lies a row of n cells numbered 1 to n from left to right. Cell i contains a positive integer c_i. One possible board of size $n = 16$ could look like this:

2	4	3	32	1	22	1	3	7	2	19	3	1	8	2	4

The rules are as follows:

1. Your character starts on the leftmost cell.
2. On each turn, you roll a fair 6-sided die; let X be the resulting number. You advance $X \times c_i$ cells, where i is the index of the cell that your character is standing on.
3. You win once you leave the board. In other words, you win when your index $i > n$.

Here is one possible playthrough using this board.



In the example above, you start at the first cell, containing the integer $c_1 = 2$. On your first turn, you roll a 2, so you advance 4 cells. Then you roll a 6 on cell 5 (with $c_5 = 1$), so you advance 6 cells. Lastly, you roll a 1 on cell 11 (with $c_{11} = 19$), advancing 19 cells, which brings you off of the board. You win! In total, you took 3 turns.

You are so bored that you start calculating the expectation of the number of turns needed to win. You reason that writing a computer program will help you do this efficiently. Given the board as input, provide a dynamic programming algorithm that runs in time $O(n)$ to calculate the expected number of turns needed to win the game.

In your solution, include a formal proof that the algorithm computes the expectation correctly.