CS 6150: HW2 – Dynamic Programming, Greedy Algorithms

Submission date: Thursday, Sep 29, 2016

This assignment has 6 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Probability of k heads	5	
Count number of parsings	5	
The Ill-prepared Burglar	10	
Central nodes in trees	10	
Faster LIS	10	
Maximizing Happiness	10	
Total:	50	

Question 1: Probability of k heads .... **[5]**

Suppose we have n different coins, and suppose the ith coin has a probability p_i of turning up heads, when tossed. Now, say we toss all the coins, and we are interested in the event that we obtain precisely k heads. Design an algorithm that computes the probability of this event in $O(n^2)$ time. [For this problem, you may assume that multiplying any two numbers in the interval $(0, 1)$ takes $O(1)$ time.]
[Source: Dasgupta et al. textbook]

Question 2: Count number of parsings .... **[5]**

Suppose we are typing on an old keyboard with a broken space key. Given a string, there can now be several ways to break it up into component words, e.g., TENTENDING, could either have been TEN TEN DING, or TENT ENDING. Given a string of length N, and a collection of all the words in the language (say there are m of them, each of length at most L), design an algorithm that counts the number of possible ways to break the string into words. What is the run time of this algorithm?

Question 3: The Ill-prepared Burglar .... **[10]**

Suppose a burglar breaks into a house, and realizes the house has far too many items to fit into his sack. He thus wishes to pick the items of most value that he can end up taking.

(a) **[2]** Not being trained in algorithms, he starts stuffing items into his sack, the most valuable first. He stops when none of the items left can fit into his back. Give an example in which this strategy is not optimal.

(b) **[2]** After obvious disappointment at his performance, the robber decides to empty his sack and pick items in decreasing order of the *ratio* value/size. Give an example in which even this strategy is not optimal.

Lost in thought, the robber loses track of time and gets caught. To dissuade future (smarter) robbers, we now wish to take the most valuable items and move them to a bank locker. Unfortunately, we find ourselves in the same situation.

(c) **[5]** Suppose the locker has size S, and suppose the house has n items, with item i having value v_i and size s_i, both of which are positive integers. Design an algorithm with running time $O(nS)$ that finds the "most valuable" collection of items (i.e., max total value), and total size $\leq S$.

(d) **[1]** Note that the *input* for the problem consists of S, and the values v_i, s_i, for $1 \leq i \leq n$. Thus the space necessary to write down the input is $O(\log S + \sum_{i=1}^{n} \log v_i + \log s_i)$. Answer YES/NO, with justification: Is the dynamic programming algorithm above polynomial in the input size?

Question 4: Central nodes in trees .... **[10]**

Let T be a rooted binary tree with n vertices, and let $1 \leq k \leq n$ be an integer. We would like to mark k vertices in T so that every vertex has a nearby marked ancestor. Formally, given a set S of marked vertices, we define $cost_S(v)$ for any $v \in T$ to be the distance (i.e., the number of hops) to the closest ancestor of v that is in S. (If no such ancestor exists, the cost is ∞, and if $v \in S$, the cost is zero.)

The "loss" of a set S is defined to be $\max_{v \in T} cost_S(v)$. Design an algorithm with $O(n^2k^2)$ running time that finds a set S of size k that minimizes the loss.
[Source: Jeff Erickson's Exercises]

Question 5: Faster LIS .... **[10]**

In class, we discussed the Longest Increasing Subsequence (LIS) problem, and an algorithm based on dynamic programming with running time $O(n^2)$ (where n is the size of the input array). Let us now see how to reduce the running time to $O(n \log n)$. Let the input array be $A[0, 1, \ldots, n-1]$, and suppose the entries are *all distinct*.

Now, let $L[i]$ denote the length of the longest increasing subsequence *starting at* position i. The idea was to start with $i = n-1$, move left, and compute the $L[i]$ values in that order. The simple algorithm we saw takes $O(n)$ time to compute the value of $L[i]$ given the values $L[j]$ for all $j > i$. The question is if we can do better.

As we move from right to left, suppose we maintain additional array $B[]$ with the following property: the jth element in B is the value of the *largest* $A[i]$ in the array we have seen so far with the property that $L[i] \geq j$. I.e., it is the largest entry in the array seen so far that has an increasing subsequence of length j starting with itself.

(a) [**4**] Prove that the entries of B are strictly decreasing.

(b) [**6**] As we move from right to left, show how we can update this array maintaining the property above, and complete the algorithm for computing the LIS. Prove that its running time is $O(n \log n)$ overall.

Question 6: Maximizing Happiness .... **[10]**

Consider Santa Claus' dilemma: there are n gifts, and n children, and each child has a non-negative value for each gift. Formally, the value of gift j to child i is given by $A_{i,j} (\geq 0)$. Santa's goal is to give one gift to each child, so as to maximize the *total value*. Formally, if child i gets the $\pi(i)$'th gift, then the total value is $\sum_{i=1}^{n} A_{i,\pi(i)}$.

(a) [**4**] Suppose Santa is in a real hurry, and he assigns gifts greedily. He starts with the children in order $(1, 2, \dots)$, and for each child, gives him/her the most valuable gift among the ones remaining.

Prove that this greedy strategy can be really bad. Concretely, give a setting in which the assignment produced by the greedy strategy is a factor 1000 worse than the *best* assignment for that setting.

(b) [**6**] Suppose Santa realizes this, and decides to use local search: he starts with some assignment in his mind, and for every pair of children, he checks to see if swapping their gifts can improve the total value. If so, he swaps, and continues this process until no such swap improves the value.

Prove that for any setting of the A_{ij}'s, the solution obtained in the end has a value at least $(1/2)$ the total value of the optimum for that setting.

Bonus [10]: Suppose Santa does a more careful local search, this time picking every *triple* of children, and seeing if there is a reassignment of gifts that can make them happier. Prove that the final solution has a value that is at least $(2/3)$ times the optimum. [This kind of a trade-off is typical in local search – each iteration is now more expensive $O(n^3)$ instead of $O(n^2)$, but the approximation ratio is better.]

Note. We will see in the coming lectures that this problem can indeed be solved (optimally) in polynomial time, using a rather different algorithm.