CS 6150: HW2 – Dynamic Programming, Greedy Algorithms

Submission date: Friday, Sep 29, 2017, 11:59 PM

This assignment has 5 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Coin change revisited	10	
Counting paths	15	
The Ill-prepared Burglar	5	
Road tripping	10	
Maximizing Happiness	10	
Total:	50	

Question 1: Coin change revisited .... **[10]**

In class, we introduced the "coin change" problem, where we have k types of coins, of denominations $d_1, d_2, \ldots, d_k$ resp., and the goal is to make change for N cents, for some positive integers $N, d_1, \ldots, d_k$.

We saw that a greedy strategy does not work, and we needed to use dynamic programming.

(a) [**3**] We discussed a naive dynamic programming algorithm that takes space $O(N)$. Show how to improve it to $O(\max_i(d_i))$.

(b) [**7**] Design an algorithm that outputs the number of different ways in which change can be obtained for N cents using the given coins. (Two ways are considered different if they differ in the number of coins used of at least one type.) Your algorithm needs to have time and space complexity polynomial in N, k.

Question 2: Counting paths .... **[15]**

We saw that the existence of Hamiltonian paths (paths of length $n-1$ with no vertex visited twice) can be determined using time/space $2^n \text{poly}(n)$.

(a) [**2**] Prove that we can also *count* the number of Hamiltonian paths using similar time and space complexity.

(b) [**8**] One could similarly ask if we can count the number of paths of length k with no vertex visited twice, for a given parameter k, in time that is roughly $2^k \text{poly}(n)$ and not n^k. This turns out to be rather tricky,[1] Meanwhile, there are graphs for which the problem is much easier.

Given a *directed acyclic graph* G on n vertices, (a directed graph with no directed cycles) give an algorithm that finds the number of directed, length-k paths between two given vertices u and v. The algorithm should run in time $\text{poly}(n, k)$.

(c) [**5**] Show how you can use the algorithm from part (b) above to solve part (b) of Question 1. (To attempt this part, you don't *need* to have solved part (b).)

Question 3: The Ill-prepared Burglar .... **[5]**

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, in decreasing order of value. He stops when none of the items left can fit into his back. Give an example in which this strategy is not optimal.

(b) [**3**] After obvious disappointment at his performance, the burglar 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.

Question 4: Road tripping .... **[10]**

Let us roll back to the days when a road trip meant writing all your favorite music onto CDs and carrying them along.

Suppose we have n songs, of durations $d_1, d_2, \ldots, d_n$, where each d_i is less than one hour. Suppose that every CD can hold one hour of music. The goal is to write all the songs to a set of CDs, so as to minimize the number of CDs required.

Consider the natural greedy algorithm for the problem: at every step, we have a list of CDs containing the music written so far (initially empty). Now consider the songs in the given order, and for song i, write it to the first CD so far that has sufficient space for the song. If there is no such CD, then we add a new CD to the list, and write song i to it.

(a) [**3**] Give an example in which the greedy algorithm is not optimal.

(b) [**7**] Prove that the greedy algorithm is always within a factor 2 of the "best possible" number of CDs. [*Hint:* You may want to compare the number of CDs used by the algorithm with the "total duration" of the songs. Also, play around with small examples to see how the algorithm is doing.]

[1] If you are interested, look for a paper of Alon, Yuster and Zwick titled "Color Coding".

Question 5: Maximizing Happiness . . . . **[10]**

Consider the weighted bipartite matching problem we saw in class: there are n gifts, and n children, and each child has a non-negative valuation for each gift. Formally, the value of gift j to child i is given by $A_{j,i}$. We assume that all $A_{j,i} \geq 0$. Santa's goal is to give one gift to each child, so as to maximize the *total value*. Formally, if gift j is given to the $\pi(j)$'th child, then the total value is $\sum_{j=1}^{n} A_{\pi(j),j}$.

(a) [**3**] We saw in class that starting with any solution and performing "swaps" as long as they improve the solution, we get to a solution whose cost is at least half that of the best possible assignment of gifts. The point of this exercise is to prove that this factor cannot be improved significantly. Give an example (i.e., an n and the matrix A) for which the cost of the locally optimal solution is at least a factor 2 smaller than the cost of the globally optimal solution. (You need to exhibit a locally optimum solution, and a globally opt one.)

(b) [**7**] Suppose we now perform a more elaborate local search, this time picking every *triple* of edges in the current solution, and seeing if there is a reassignment of gifts between the end points of these edges that can improve the total value. Prove that a locally optimal solution produced this way has a value that is at least (2/3) the optimum value. [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.