CSE 202: Theory Problems

https://sites.google.com/view/cse202w17

January 22, 2017

Overview

In most theory problems, your goal is to design an algorithm, to prove its correctness, and to prove an upper bound on its running time. It is recommended that your solution contains three sections. In the first section, you describe an algorithm. Make sure to provide a description of your algorithm and its underlying ideas. If needed, provide a pseudocode, but don't provide just a pseudocode, without explaining what is going on in it (just a pseudocode will be given zero credit). In the second question prove formally that your algorithm solves the given computational problem correctly. In the third section, prove formally an upper bound on the running time of your algorithm.

Note that in contrast to code problems, in theory problems instead of constraints on the input data you are given an asymptotic upper bound on the running time of an algorithm. For theory problems, the input format is not so important as in code problems. In your solution, you may safely assume that all the input data is already read. Also, in theory problems, we don't care about edge cases, integer overflow, stack overflow, and other practical issues.

Each theory problem is worth 10 points. The rubrics are going to be roughly as follows: 4 point for the correct algorithm, 3 points for correctness, 3 points for an upper bound. Of course, a slow algorithm will be given less than 7 points even if you describe it well and prove that it is correct.

Contents

1 Introduction

1.1 Sum of squares of Fibonacci numbers

Description. Recall that Fibonacci numbers are defined recursively as follows: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Prove by induction on $n \geq 1$ that

$$\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}\,.$$

Instructions. Submit your solution on a piece of paper before the beginning of the lecture on **Tue Jan 17** (that is, at 1:59PM at the latest).

Solution. **Base case.** The statement holds for $n = 1$: $F_1^2 = 1 = F_1F_2$.
Induction step. Assume that $n \geq 2$ and that the statement holds for $1, 2, \ldots, n$. Let's prove it for $n+1$:

$$F_1^2 + F_2^2 + \cdots + F_{n+1}^2 = F_nF_{n+1} + F_{n+1}^2 = F_{n+1}(F_n + F_{n+1}) = F_{n+1}F_{n+2}.$$

2 Greedy Algorithms

2.1 Changing money

Description. The goal of this problem is to design a simple greedy algorithm for changing money optimally.

Input. A non-negative integer m.

Output. The minimum number of coins needed to change m into coins with denominations 1, 5, and 10.

Running time. $O(m)$.

Sample.

Input. 28.

Output. 6.

Explanation. $28 = 10 + 10 + 5 + 1 + 1 + 1$.

Remark. Note that it is crucial for the analysis of the algorithm that available denominations are 1, 5, and 10. If they were, say, 1, 4, and 6, then the corresponding greedy algorithm would not be optimal: it would change 8 with three coins $(6 + 1 + 1)$, while it is possible to change it with just two coins $(4 + 4)$.

Instructions. Submit your solution on Schoology. Make sure to describe your algorithm (don't just provide a pseudocode!), to prove that it always finds the minimum number of coins, and that it does so in time $O(m)$.

2.2 Organizing a party

Description. Alice wants to throw a party and is deciding whom to call. She has n people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least two other people whom they know and at least two other people whom they don't know.

Input. A list of pairs of people who know each other. Assume that the people are numbered from 1 to n.

Output. The maximum number of people that can be invited to the party.

Running time. $O(n^3)$.

Sample.

Input. $(1,5)$, $(1,6)$, $(1,8)$, $(2,4)$, $(2,7)$, $(2,8)$, $(3,8)$, $(4,7)$, $(4,8)$, $(5,6)$, $(5,8)$, $(6,8)$, $(7,8)$.

Output. 6.

Explanation. There are eight people in this case. We may invite $1,2,4,5,6,7$: 1 knows $5,6$ and does not know $2,4,7$; 2 knows $4,7$ and does not know $1,5,6$, and so on. At the same time 3 and 8 cannot be invited: 3 knows only one person (8) while 8 knows everybody.

Instructions. Submit your solution on Schoology. Make sure to describe your algorithm (don't just provide a pseudocode!), to prove that it is correct (i.e., always finds the maximum number of people) and has the required running time.

3 Divide and Conquer

3.1 Fixed point in an array

Description. The goal in this problem is to check whether the given sorted array $A[1\dots n]$ of pairwise distinct integers contains a fixed point, i.e., an index i such that $A[i]=i$.

Input. A sorted array $A[1\dots n]$ of pairwise distinct integers. I.e., for all $1 \le i \ne j \le n$, $A[i] \ne A[j]$, and $A[1] < A[2] < \cdots < A[n]$.

Output. Output 1 if there is an index i such that $A[i]=i$, and 0 otherwise.

Running time. $O(\log n)$.

Sample 1.

Input. $1,2,3$.

Output. 1.

Explanation. Each element is a fixed point in this case: $A[1] = 1$, $A[2] = 2$, $A[3] = 3$.

Sample 2.

Input. $-4, -2, 4, 5, 8$.

Output. 0.

Explanation. $A[1] = -4 \neq 1$, $A[2] = -2 \neq 2$, $A[3] = 4 \neq 3$, $A[4] = 5 \neq 4$, $A[5] = 8 \neq 5$.

Remark. In this problem we assume that the input array is already read and your algorithm just needs to process it in time $O(\log n)$ to find out the answer (i.e., you algorithm does not need to spend $\Theta(n)$ time for reading the input).

3.2 Organizing a lottery

Description. You are organizing an online lottery. To participate, a person bets on a single integer. You then draw several ranges of consecutive integers at random. A participant's payoff then is proportional to the number of ranges that contain the participant's number minus the number of ranges that does not contain it. You need an efficient algorithm for computing the payoffs for all participants. A naive way to do this is to simply scan, for all participants, the list of all ranges. However, you lottery is very popular: you have thousands of participants and thousands of ranges. For this reason, you cannot afford a slow naive algorithm.

Input. A sequence of n points $x_1, \dots, x_n$ on a line and a sequence of m segments $[a_1, b_1], \dots, [a_m, b_m]$ on a line.

Output. For each point x_i, output the number of segments it is contained in, i.e., the number of segments $[a_j, b_j]$ such that $a_j \le x_i \le b_j$.

Running time. $O((n+m)\log(n+m))$.

Sample.

Input. Points: $1, 6, 11$. Segments: $[2, 3]$, $[0, 5]$, $[7, 10]$.

Output. $1, 0, 0$.

Explanation. Here, we have two segments and three points. The first point lies only in the first segment while the remaining two points are outside of all the given segments.

3.3 Finding a peak in sublinear time (advanced)

Description. Given an $n \times n$ matrix containing pairwise different integers, you goal is to find a *peak* in it in time $o(n^2)$. An element is called a peak, or a local maximum, if it is strictly greater than all its adjacent neighbors (an interior element of the matrix has four neighbors, an element on the border has three neighbors, a corner element has two neighbors). Each matrix has at least one local maximum since a global maximum is also a local maximum. In general, a matrix may contain many different local maximums. Your goal is to find at least one such local maximum.

Input. A matrix $A \in \mathbb{N}^{n \times n}$ consisting of pairwise different integers.

Output. An element $A[i,j]$ that is larger than all its neighbors.

Running time. $o(n^2)$.

Sample 1.

Input. $\begin{bmatrix} 21 & 17 & 92 \\ 31 & 22 & 29 \\ 14 & 76 & 10 \end{bmatrix}$

Output. 31.

Explanation. Other local maximums in this matrix: 92, 76.

Sample 2.

Input. $\begin{bmatrix} 31 & 10 & 41 & 42 & 43 \\ 32 & 11 & 40 & 17 & 44 \\ 33 & 12 & 39 & 18 & 45 \\ 34 & 13 & 38 & 19 & 46 \\ 35 & 36 & 37 & 20 & 47 \end{bmatrix}$

Output. 47.

Explanation. There is just one local maximum in this case.

Sample 3.

Input. $\begin{bmatrix} 11 & 24 & 23 & 22 \\ 12 & 25 & 30 & 21 \\ 13 & 26 & 29 & 20 \\ 14 & 77 & 28 & 19 \\ 15 & 16 & 17 & 18 \end{bmatrix}$

Output. 30.

Explanation. Another local maximum: 77.

Remark. As with the first theory problem in this module, we assume here that the matrix is already loaded. Our goal is to find a local maximum in *sublinear* time since the size of the input in this case is n^2. That is, we would like to find the answer by accessing only a small fraction of the elements of the matrix.

Recall that this is in advanced problem. It is not so difficult to find an algorithm for this problem, but proving that it is correct is tricky.

4 Dynamic Programming

4.1 Planning a trip

Description. You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty–that is, the sum, over all travel days, of the daily penalties.

Input. Sorted sequence $a_1 < a_2 < \cdots < a_n$.

Output. The optimal sequence of hotels at which to stop.

Running time. $O(n^2)$.

Sample.

Input. 205, 301, 410, 500.

Output. 205, 410, 500.

Explanation. The total penalty of the trip $0 \to 205 \to 410 \to 500$ is $5^2 + 5^2 + 90^2$. To compare, the penalty of the trip $0 \to 205 \to 301 \to 500$ is $5^2 + 104^2 + 1^2$.

4.2 Partitioning souvenirs

Description. You and your two friends have been trekking through various far-off parts of the world and have accumulated a big pile of souvenirs. At the time you weren't really thinking about which of these you were planning to keep and which your friends were going to keep, but now the time has come to divide everything up.

Input. A sequence of positive integers $x_1, x_2, \dots, x_n$ (representing the values of souvenirs).

Output. Output 1 if it is possible to partition them into three sets with equal sums, and 0 otherwise. More formally, we would like to check whether there exist three disjoint sets $S_1, S_2, S_3 \subseteq \{1, \dots, n\}$ such that $S_1 \cup S_2 \cup S_3 = \{1, \dots, n\}$, and $\sum_{i \in S_1} x_i = \sum_{j \in S_2} x_j = \sum_{k \in S_3} x_k$.

Running time. $O(nX^2)$ where where $X = x_1 + x_2 + \cdots + x_n$.

Sample 1.

Input. 7, 7, 7.

Output. 1.

Sample 2.

Input. 9, 10, 11, 12, 12.

Output. 0

Sample 3.

Input. 1, 2, 3, 4, 5, 5, 7, 7, 8, 10, 12, 19, 25.

Output. 1.

Explanation. $1 + 3 + 7 + 25 = 2 + 4 + 5 + 7 + 8 + 10 = 5 + 12 + 19$