CSCI261 Analysis of Algorithms, Fall 2020/21, Homework 3

Due Friday, October 2, 2020, 11:59pm

Problem 1

In a little town the roads are all laid out in a perfect grid. There is a North-South road every block. There is an East-West road every block. At every intersection, there is a traffic light. In order to make sure people obey the traffic lights, there are police cars stationed at some of the intersections. You have decided to open a corner donut store at one of the intersections, and need to identify the most profitable location for your store. The most profitable location will be the one that minimizes the sum of the distances that the traffic police have to travel to reach your store. The police must travel by using the existing roads (they can not travel diagonally).

You are given a collection of integer coordinates $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ that represent the locations of traffic police. Give an $O(n)$ algorithm that determines the best corner location (given by (x_{best}, y_{best})) for your donut store. That is, (x_{best}, y_{best}) should be chosen such that $\sum_{i=1}^{n} |x_{best} - x_i| + |y_{best} - y_i|$ is as small as possible, and (x_{best}, y_{best}) should also be integer coordinates.

Problem 2

Given is a sequence of numbers $a_1, a_2, a_3, \ldots, a_n$. We want to cross out the smallest number of elements from this sequence so that the remaining elements are listed in increasing order. This problem is known as the *longest increasing subsequence* problem and can be described formally as follows: We say that $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$ is a *subsequence* of the given sequence if and only if $1 \leq j_1 < j_2 < \ldots < j_k \leq n$. A sequence of numbers $b_1, b_2, \ldots, b_m$ is *increasing* if and only if $b_1 < b_2 < \ldots < b_m$. In the *longest increasing subsequence* problem we search for an increasing subsequence of $a_1, a_2, a_3, \ldots, a_n$ that is the longest possible (i.e, k is as large as possible).

For example, for sequence $8, 2, 5, 7, 3, 4, 9, 6, 10$, the longest increasing subsequences are $2, 3, 4, 6, 10$ and $2, 5, 7, 9, 10$ – either of them is a valid longest increasing subsequence.

Mr. Brilliant came up with the following greedy approaches to the problem:

- Find the smallest number in the sequence – if there are more than one, select the one with the smallest index. Suppose the number is a_ℓ. Output the number. Repeat the previous steps for the input sequence $a_{\ell+1}, a_{\ell+2}, \ldots, a_n$, until the input sequence is empty.
- Try the following with $\ell = 1, 2, \ldots, n$: Let $i = \ell$. Include a_i in the subsequence. Find the smallest j such that $j > i$ and $a_i < a_j$. If there is no such j, stop. Else, let $i = j$ and repeat the steps described in the last two sentences. Once you are done trying all possible ℓ's, out of the n obtained subsequences output one with the longest length.

For every greedy approach:

(a) Give a corresponding detailed and properly structured pseudo code. In particular, the pseudo code should be like code, without worrying about syntactical issues like semicolons. Do not use any go-to statements, breaks, continues, and exceptions.

(b) Estimate the running time of your pseudo code using big-Oh notation and reason the estimate.

(c) Determine whether the approach works. Does it correctly identify a longest subsequence for every input? If yes, argue why it always works. If not, provide a counterexample. In particular, provide 1. the input sequence on which the approach fails, 2. an optimal solution (that is, a longest possible increasing subsequence for the provided input), and 3. the solution produced by the greedy approach (i.e., the sequence produced by the greedy approach on this input – this sequence should be shorter than the one in 2.).

Problem 3

This problem is also about the longest increasing subsequence problem (see Problem 2). You will implement a recursive approach and a dynamic-programming-based approach and compare their running times. In both cases we are interested only in finding the length of the sequence, not the sequence itself.

- Implement the following recursive approach. Implement the function `incrSubseqRecursive`(j, A), that computes the maximum length of an increasing subsequence of the sequence $a_1, a_2, \ldots, a_n$ (stored in the array A) that ends with the element a_j. This function tries to find a previous element a_i such that $i < j$ and $a_i < a_j$, and then it recursively searches for the maximum length of an increasing subsequence of $a_1, a_2, \ldots, a_i$. It tries up to $j-1$ different i's and it chooses the one that gives rise to the maximum length. By concatenating a_j to the subsequence of $a_1, a_2, \ldots, a_i$ ending with a_i, we get a longest increasing subsequence of $a_1, a_2, \ldots, a_j$.

- Implement a bottom up dynamic programming approach (see lecture slides).

- Generate about 10 inputs for different values of n (how large can n be?) and run your implementation on these inputs. Plot the actual execution time (y-axis) for the different values of n (x-axis). (Alternatively, you can provide this information in a table.) Then include a short paragraph describing your findings.