Algorithm Design and Analysis

Assignment 4

Deadline: Dec 9, 2024

For all the dynamic programming exercises, please 1) clearly define the objective function, 2) write the recursive formulation of the objective function, and 3) write the pseudocode of your algorithm that implements the recursive formulation.

1. (25 points) Let $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_m$ be strings. A string $C = c_1 \cdots c_{m+n}$ is called a *shuffle* of A and B if there are two increasing functions $\phi : [n] \to [n+m]$ and $\psi : [m] \to [n+m]$ (we use $[k]$ to denote the set $\{1, \ldots, k\}$ for a positive integer k) such that $A = c_{\phi(1)} \cdots c_{\phi(n)}$ and $B = c_{\psi(1)} \cdots c_{\psi(m)}$ and $\phi(i) \neq \psi(j)$ for every i, j. Design a polynomial-time algorithm that, given strings A, B, C, decide if C is a shuffle of A and B.

2. (25 points) Given n gifts located on a $(m \times m)$ grid. The i-th gift is located at some point (x_i, y_i) (integers chosen in $1 \cdots m$) on the grid, with value $v_i \geq 0$. A player at $(1, 1)$ is going to collect gifts by several *Upper-Right Move*. In particular, assuming the player is currently located at (x, y), he can make one *Upper-Right Move* to another point (x', y') where $x' \geq x$ and $y' \geq y$. The cost of this movement is $(x' - x)^2 + (y' - y)^2$. The player will collect the i-th gift when he is at point (x_i, y_i). There is no restriction for the number of *Upper-Right Move* and the final location of the player.

   (a) (15 points) Design an $O(m^2)$ time algorithm to maximize the player's profit, i.e., the sum of value he collects minus the sum of cost he pays for his *Upper-Right Move*. Prove the correctness of your algorithm.

   (b) (10 points) Sometimes, n can be much smaller than m. Can you design another algorithm that runs in $O(n^2)$ time for this situation? Prove the correctness of your algorithm.

   (c) (0 points) (Just for fun) What if the player is not restricted to Upper-Right Moves? Can we still design polynomial-time algorithms? You are welcome to just guess the answer.

3. (25 points) In the *subset-sum problem*, you are given a set $T = \{a_1, \ldots, a_n\}$ of n positive integers and a positive integer k as inputs, and you are to decide if there is a subset S with sum exactly k. Notice that a set in this problem may contain multiple copies of an integer.

   (a) (10 points) Design an $O(kn)$ time algorithm for this problem. Note: This is not a polynomial time algorithm. In fact, the subset-sum problem is a well-known NP-complete problem, as the partition problem you have seen in the class is a special case of this problem with $k = \frac{1}{2}\sum_{i=1}^{n} a_i$ and the partition problem is NP-complete.

   (b) (15 points) Suppose now you are guaranteed that there exists a subset S with sum exactly k and you are given an extra input parameter $\varepsilon > 0$. Design an algorithm to find a subset S' such that

$$\sum_{a_i \in S'} a_i \in [(1-\varepsilon)k, (1+\varepsilon)k] .$$

   Your algorithm's running time should be polynomial in terms of $1/\varepsilon$ and n. Prove the correctness of your algorithm, and analyze its running time.

4. (25 points) Let G be a tree with n vertices. We have seen in the class how to use dynamic programming to find the maximum independent set in G. In this problem, we work on *counting*. We assume that it takes $O(1)$ time to store and multiply two integers.

   (a) (10 points) Design an $O(n)$ time algorithm to count the number of independent sets in G. Prove the correctness of your algorithm and analyze its time complexity.

   (b) (15 points) Design an efficient algorithm to count the number of *maximum* independent sets in G. Prove the correctness of your algorithm and analyze its time complexity.

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.