15-451/651 Algorithm Design & Analysis, Spring 2024

Recitation #5

Objectives

- Practice dynamic programming algorithms (using small steps).
- Understand how to analyze the complexity of a dynamic programming algorithm.
- Understand and practice common strategies for defining subproblems, e.g.:
  - Considering a **prefix** (first i elements) of the input (*e.g. Knapsack*).
  - Consider a **substring** $i \dots j$ of the input (*e.g., Optimal BSTs in 210*).
  - Consider **smaller values** of the input parameters (*e.g., Knapsack*).
  - Consider **subtrees** (Tree DP) (*e.g., Max-weight independent set on a tree*)
  - Consider **subsets** of the elements of the input (*e.g., TSP*).
  - Remember most **recent decision** / add more information (*e.g, final vertex in TSP*).

Recitation Problems

1. **(Vacuums)** Suppose that you are a door-to-door salesman, selling the latest innovation in vacuum cleaners to less-than-enthusiastic customers. Today, you are planning on selling to some of the n houses along a particular street. You are a master salesman, so for each house i, you have already worked out the amount c_i of profit that you will make from the person in the house if you talk to them. Unfortunately, you cannot sell to every house, since if a person sees you selling to their neighbor, they will hide and not answer the door for you. Therefore, you must select a subset of houses to sell to such that none of them are next to each other, and such that you make the maximum amount of money.

   For example, if there are 10 houses and the profits that you can make from each of them are 50, 10, 12, 65, 40, 95, 100, 12, 20, 30, then it is optimal to sell to the houses $1, 4, 6, 8, 10$ for a total profit of \$252. Devise a dynamic programming algorithm to output the maximum profit you can make.

   (a) Define a good subproblem for your DP.
   (b) Write the base cases of your recurrence.
   (c) Write the recurrence cases to finish your DP recurrence.
   (d) Finish the problem by explaining how to run your recurrence.
   (e) What is the time bound for this DP?

2. **(Tidying Up)** You are in a room containing n objects at integer coordinates (x_i, y_i) for $1 \leq i \leq n$. In the center of the room at $(0,0)$, you have a box in which you would like to place all of the objects. The problem is that the objects are quite heavy, so you can only manage to carry at most *three* of them at a time! The box is also heavy, so you can not move the box. The time that it takes you to move between two points i and j is $|x_i - x_j| + |y_i - y_j|$.

   Assuming that you start at position $(0,0)$ with the empty box, what is the minimum amount of time required for you to place all of the objects into the box? Give an algorithm for this problem that runs in $O(2^n \cdot n^2)$ time.

   Hints:

   - Any solution to this problem will essentially be a bunch of trips away from the box to fetch items, and then back to the box to put them away. Does the relative ordering of which trip happens first matter? [No.]

3. **(Cheapest Tree Separation)** There are N cities, numbered from 1 through N, connected by $N-1$ roads, forming a weighted tree. Countries A and B each occupy a set of cities (no city is occupied by both countries, and some cities may not be occupied at all).

   To stop fighting between the two countries, you want to destroy roads such that no city occupied by country A is connected to a city in country B. Destroying a road of length x costs x dollars. What is the minimum cost required? Given a linear time algorithm.

4. **(Number of Increasing Partitions (Optional))** We are given an array $A = [A_1, A_2, \ldots, A_n]$. How many ways are there to partition A into contiguous, non-overlapping subarrays such that the **maximum** element in each subarray are nondecreasing from left to right?

   For example, if $A = [1,2,2,1,3,2,1]$ then the partition $[1,2],[2,1],[1,3,2,1]$ is valid because the maximum element of each subarray is 2, 2, 3 and these are nondecreasing from left to right. However, the partition $[1,2,2,1,3],[2,1]$ is NOT a valid partition because the maximum of the first subarray (3) is greater than the maximum of the second subarray (2).

   (a) Define a set of subproblems to which we can apply dynamic programming.

   (b) Give a recurrence relation with base cases that show how to compute the value of a subproblem

   (c) Show that your dynamic programming solution can be evaluated in $O(n^2)$ time, then show that it can be improved to $O(n)$ time