15-451/651 Algorithm Design & Analysis

Fall 2022, Recitation #5

Objectives

- Practice dynamic programming algorithms for sequences and trees (using small steps).
- Understand how to analyze the complexity of a dynamic programming algorithm.
- Understand and practice common strategies for defining subproblems.
  - Considering a prefix (first i elements) of the input (*e.g., LCS, 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 next lecture*).
  - Remember most recent decision, and strengthen recurrence (*e.g, TSP next lecture*).

Recitation Problems

1. **(Longest Common Alignment?)**

   Recall the **Longest Common Subsequence (LCS)** problem from lecture: we want to find the longest sequence of characters that appear left-to-right in two strings ($s, t \in \Sigma^*$ for some alphabet Σ). This problem can be solved using dynamic programming using the following recurrence.

$$\text{LCS}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max(\text{LCS}(i-1,j), \text{LCS}(i,j-1)) & s[i] \neq t[j] \\ 1 + \text{LCS}(i-1,j-1) & s[i] = t[j] \end{cases}$$

   Let's generalize this problem slightly.

   For **(LCS)**, we only add 1 to our recurrence if two characters *match*, and 0 otherwise.

   Suppose that instead of adding 0 or 1 based on whether the characters match, we define a *score function* $A : \Sigma \times \Sigma \to \mathbb{Z}$ that takes two characters and outputs a integer representing the match "quality". For example, if we wanted (a, a) to "full match", but (a, b) to "half match" and (a, c) to "not match", then we would let $A(a, a) = 2$, $A(a, b) = 1$, and $A(a, c) = 0$.

(a) Given such a score function, modify the above recurrence so that it returns the score of the subsequences with the maximum score (highest "quality") over all characters.

(b) What would be the scoring function if we want our recurrence to mimic **(LCS)**?

2. **(Swap To The Top)**

An *ascending binary string* is a binary string $b = b_1 b_2 \cdots b_n \in \{0,1\}^n$ such that $b_1 \leq b_2 \leq \cdots \leq b_n$. Examples of ascending binary strings in $\{0,1\}^4$ include 0011, 0111, 0000 and non-ascending binary strings include 1000, 1010, 0101.

Consider the `blockflip` operation on binary strings: a $\texttt{blockflip}(b, i, j)$ on a binary string b inverts the bits between indices i and j, and only i and j. For example,

$$\texttt{blockflip}(011001, 2, 4) = 001101$$
$$\texttt{blockflip}(1100, 1, 4) = 0011$$

We want to write a dynamic programming algorithm which accepts an arbitrary binary string b and calculates the minimum number of `blockflip` operations required to make b an ascending binary string.

(a) Suppose we had two ascending binary strings, ℓ and r.

At most how many `blockflip` operations are required to make the string $b = \ell r$ ascending (ℓ concatenated with r)?

(b) With this observation, how should we define a subproblem? How many of them are there?

(*Hint:* use one of the techniques from **Objectives**)

(c) Given answers to our subproblems, we now need to find the minimum number of `blockflip` operations for a given string. Write a recurrence that calculates the minimum number of `blockflip` operations to make a binary string b ascending given answers to our subproblems.

(d) What are the base case(s) for our recurrence?

(e) What is the time complexity of our dynamic programming algorithm?

(*Hint*: count the number of subproblems, and multiply this by the amount of work we do at each subproblem; don't forget the base case(s)!).

3. **(Green and Blue)**

We have a binary tree T with n nodes, some of which are colored green and the rest blue. All the nodes have been assigned **positive** integer weights. We also have a nonnegative integer k. Using this, we would like to find the (connected) subgraph with highest weight using at most k green nodes.

(a) Consider some unknown (non-empty) binary tree $T' = (L, (c, x), R)$ where L and R are the left and right subtrees, respectively. c is the color (green/blue), and x is the weight of the node.

Let's try to find a good subproblem definition for the tree. Say we define $W(T', k')$ to be the maximum weight connected subgraph of T' using at most k' nodes. What might be the problem with using this as a subproblem, and how can we fix it?

(b) What are the base cases for this problem?

(c) Use the base cases to determine a recurrence for this problem. It may be helpful to case on whether the root is green or blue.

(d) What is the time complexity of this algorithm in terms of n and k?

(*Hint*: count the number of subproblems, and multiply this by the amount of work we do at each subproblem)

Further Review

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, you have already worked out the amount c_i of profit that you will make from the person in house i if you talk to them. Unfortunately, you cannot sell to every house, since if a person's neighbour sees you selling to them, 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 solve this problem.

   (a) Define a set of subproblems that we could use to apply dynamic programming

   (b) What are the base case subproblems and what are their values?

   (c) Write a recurrence relation that describes the solutions to the subproblems

2. **(Shortest Common Supersequence)** Consider a pair of sequences $a_1, ..., a_n$ and $b_1, ..., b_m$ of length n and m. A supersequence of a sequence a is a sequence that contains a as a subsequence. Devise an algorithm that finds the length of a shortest common supersequence of a and b. A common supersequence is a sequence that is both a supersequence of a and of b. Your algorithm should run in $O(nm)$.

3. **(Matrix)** You are given a positive integer N and a 26 by 26 matrix A whose entries are either 0 or 1. How many strings of length N, consisting of lowercase English letters, are there such that for all $1 \leq i, j \leq 26$ where $A_{ij} = 0$, character j does not appear directly after character i?

   (a) Define a suitable set of subproblems to apply dynamic programming

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

   (c) Argue that your dynamic program solves the problem in $O(N)$ time

4. **(Number of Increasing Partitions)** Define the value of an array to be the value of its maximum element. We are given an array $A = [A_1, A_2, \ldots, A_n]$. How many ways are there to partition A into subarrays, or "pieces", such that the values of the pieces, from left to right, are nondecreasing? For example $A = [1, 2, 1, 3, 2, 1]$ has a valid partition $[1, 2], [1, 3, 2, 1]$, but $[1, 2, 1, 3], [2, 1]$ is an invalid partition.

   (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

5. **(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.

   (a) Give a set of suitable subproblems to which we can apply dynamic programming

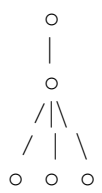
   (b) Give a recurrence relation with base cases that computes the value of each subproblem

   (c) Argue that your dynamic program solves the problem in $O(n)$ time

6. **(Connected Subgraphs of a Tree)** You're given a rooted tree T of n nodes, along with a constant k. The goal is to count the number of connected subgraphs of T of size k in $O(nk^2)$ time.

   Use the following notation. Let n_i be the number of children at node i, and let $c_{i,1}, c_{i,2}, \ldots, c_{i,n_i}$ denote the children of node i.

   For example, if the tree is



   and $k = 3$ then the answer is six.

   Hint: First try to solve the case when each node of the tree has at most two children. Then extend your solution to the general case.

7. **(Bin Packing)** You are given a collection of n items, and each item has size $s_i \in [0, 1]$. You have many bins, each of unit size, and you want to pack the n items into as few bins as possible. (Each bin can take a subset of items, whose total size is at most 1.)

   Show that you can solve this problem in time $O(4^n)$. Then improve this bound to $O(3^n)$.