CMSC 37000: Homework 4

Caleb Derrickson

March 7, 2024

Contents

Problem 1

You work for a commuter rail agency in city R. The agency wants to open coffee shops at railroad stations, and you need to design a program that finds the most economical way to do that.

The rail network is represented by a tree $T = (V, E)$ rooted at vertex R. There is a train route between R and every leaf of T. the agency wants to open coffee shops at some stations so that there is a coffee shop at at least one of every k consecutive stations on every route. The cost of opening a coffee shop at station u is $c_u > 0$. Your goal is to design a DP-algorithm that finds the most economical way to open the coffee shops. The algorithm should run in polynomial time. You don't need to prove the correctness of your algorithm.

Solution:

The problem at hand can be broken down into the following subproblems: we wish to find the minimum cost of opening a new coffee shop of a subtree rooted at vertex u, where we ensure that we between any two stops of k distance, we have at least one coffee shop. Under these conditions, we consider creating a DP table, M[u, i], where the first index u represents any vertex of the tree T, and i is the distance from the previous coffee shop we have opened.

The initialization of our DP table considers any u which is a leaf of T. To this extent, we assign the value 0 for any u which is a leaf.

Next, we write the recurrence formula for our algorithm. We first consider a vertex u of T which has some number of children, which we will denote K_u. Note that, since u is not a leaf, $|K_u| > 0$. Next, we need to consider the cases when $i = k$ and $i < k$. When $i = k$, we should include this value into our DP table, as well as its children. Thus,

$$M[u, i] = c_u + \sum_{v \in K_u} M[v, 0], \quad \text{for } i = k.$$

If $i < k$, but we deem it prudent to open a shop at this point (due to it being a lucrative investment), we consider taking the minimum of either opening the shop here, or moving to the next point. To this end, we have

$$M[u, i] = \min\{c_u + \sum_{v \in K_u} M[v, 0], \sum_{v \in K_u} M[v, i + 1]$$

For each step in the recurrence, we consider either adding a shop, setting $i = 0$, then moving on; or moving onto the next stop entirely. The recurrence then calculates the minimum cost of doing such an operation, thus leaving us overall with a value at the end of the table with minimum cost for the route. Therefore, the recurrence is given.

In order to compute the running time, we should consider how many iterations our algorithm takes. Supposing there are $|V|$ vertices in the tree, our algorithm will consider k possible values for i. Since we also need to traverse the tree as well, we consider the number of edges in our graph as well. Therefore, our algorithm will run in $O(k(|V| + |E|))$ time.

Problem 2

Consider the following puzzle. There are n chips in front of you arranged in a line. Each chip is colored with one of k colors. Your goal is to remove all of the chips in the minimum possible number of steps. At each step, you can close the created gap by moving the remaining chips together. Design a polynomial-time algorithm that finds an optimal solution to this puzzle. Prove your algorithm is correct.

Solution:

This problem can be broken into the following subproblems: choosing the optimal consecutive sequence of points as to remove all chips of a particular color c from the set of chips. To this end, we define the the table as T[i, j, c], where $1 \leq i \leq j \leq n$, is the starting and ending position of our proposed subsequence to remove, and c denotes the color in our color set of the subsequence. Note that c should be only one color for each removed subsequence.

Next, we state the initialization of our algorithm. First, we intialize T[i, j, c] = 0, for any point i > j. Next, if i = j, then we initialize T[i, j, c] = 1, where c denotes the color of chip i. Finally, if i < j, and we suppose that all chips in the subsequence form i to j are the same color, then T[i, j, c] = 1.

The recurrence formula is as follows:

$$T[i, j, c] = \min\{T[i, k, c] + T[i, k + 1, c] - 1, T[i, k, c'] + T[i, k + 1, c]\}$$

where k is the variable we are minimizing over, which has bounds $i \leq k \leq j$, and c' is any color in our color set other than c.

Let's first establish some notation:

- T[i, j, c]: This represents the minimum number of steps needed to remove all chips of color c from the subsequence starting at position i and ending at position j.
- T[i, k, c]: This denotes the minimum number of steps needed to remove chips of color c from position i to an intermediate position k.
- T[k + 1, j, c']: This signifies the minimum number of steps required to remove chips of color c' from the next position k + 1 to position j.
- The '-1' correction accounts for double-counting: When chips are removed from position i to k and from k + 1 to j, the chip at position k is counted twice. Subtracting 1 corrects for this.
- T[i, k, c']: This is the minimum number of steps required to remove chips of color c' from position i to k.
- T[i, k + 1, c]: This indicates the minimum number of steps required to remove chips of color c from position i to k + 1.

Now, we consider the recurrence formula, which aims to find the optimal solution for the main problem T[i, j, c] by considering two subproblems:

- Removing chips of color c from position i to an intermediate position k, where T[i, k, c] represents the minimum steps needed to achieve this.
- Removing chips of color c' from k + 1 to j, where T[k + 1, j, c'] represents the minimum steps needed to achieve this, with c' being a color other than c.

The formula recursively decomposes the main problem into these two subproblems: removing chips from i to k and removing chips of color c' from k + 1 to j. It then considers all possible combinations of k and c' to find the pair that minimizes the total number of steps. With the correction of '-1', the chip at position k is no longer counted twice.

Therefore, the recurrence formula embodies the recursive approach of solving the main problem by breaking it down into smaller subproblems. It then seeks the optimal combination of solutions for these subproblems to minimize the overall number of steps, making it a correct and effective strategy.