CSE 202: Design and Analysis of Algorithms **(Due: 10/26/19)**

Homework #2

Instructor: Ramamohan Paturi *Name:* Shihan Ran, *Netid:* A53313589

Problem 1: Maximum Length Chain of Subwords

Problem Description

Input: We are given a set of n distinct strings of length at most k over a finite alphabet σ.

Output: A sequence of strings that form a chain under the (consecutive) subword relation; i.e., if the output is $w_1, w_2, \ldots, w_t$ then we can write $w_{i+1} = uw_iv$ for some strings u, v.

Find a chain of maximum length.

Solution

(High-level description)

We use dynamic programming to solve this problem.

Subproblem: We refer $T(n)$ as the maximum length of chain containing w_n.

Base Case: For a set of only one string, the solution is simply the string itself.

Iteration: We can have the state transition function as follows:

$$T(n) = \max(T(i) + 1), \text{ where } w_n = uw_iv, 1 \leq i < n$$

Since our algorithm should output a sequence of string chain. We need to store some additional values:

1. $maxLength$: the length of the maximum chain
2. $maxIndex$: the tail string's index of the maximum length chain
3. $P(n)$: the index i that maximize $T(n)$

We update $maxLength$ and $maxIndex$ each time after we computed $T(n)$. With $maxLength$, $maxIndex$ and $P(n)$, we can backtrack our maximum length chain of subwords following the order

$$[w_{maxindex}, w_{P(maxindex)}, w_{P(P(maxindex))}, \ldots]$$

To find out whether $w_n = uw_iv$, we need to define a new algorithm called $isSubsequence$. We can choose a pattern matching algorithm to do the string comparison. Here, we choose KMP algorithm and the time complexity is proportional to the length of the string, which is O(k).

(Correctness)

Base Case: A set of only one string trivially has only one option for its maximum length chain of subwords.

Induction Hypothesis: Suppose all subproblems are correctly computed up to an arbitrary length of k.

Induction Step: We would like to prove that if the induction hypothesis holds, then our algorithm's "iteration" component correctly computes the subproblem of the length of $k + 1$. The following chain of logic should be sufficient:

1. Per the hypothesis, all the lengths smaller than k are correct.

2. For string w_{k+1}, we have $T(k + 1) = \max(T(i) + 1), \text{ where } w_{k+1} = uw_iv, 1 \leq i < k + 1$. We exhaustively check relevant subproblems.

3. We choose the $T(i)$ that yields a max $T(k + 1)$. Therefore, we choose the right option from among our previous calculated values and record the index i as $P(k + 1) = i$.

4. Thus, given that our hypothesis holds, our update correctly computes the length of $k + 1$'s solutions.

Because our base case, hypothesis, and step are correct, our algorithm is proven correct with a proof by induction.

(Time complexity)

The $T(n)$ loop will traverse n strings, which takes $O(n)$ time. At each $T(n)$, we will iterate through $i, 1 \leq i < n$ elements, takes $O(n)$ time. When at each w_i, we do a KMP algorithm to find out whether w_i is a subword of w_n, which will take O(k) time, and finding $\max(T(i) + 1)$ will only take $O(1)$.

Overall, our algorithm takes $O(kn^2)$.

Problem 2: Business plan

Problem Description

Consider the following problem. You are designing the business plan for a start-up company. You have identified n possible projects for your company, and for, $1 \leq i \leq n$, let $c_i > 0$ be the minimum capital required to start the project i and $p_i > 0$ be the profit after the project is completed. You also know your initial capital $C_0 > 0$. You want to perform at most k, $1 \leq i \leq n$, projects before the IPO and want to maximize your total capital at the IPO. Your company cannot perform the same project twice.

In other words, you want to pick a list of up to k distinct projects, $i_1, \ldots, i_{k'}$ with $k' \leq k$. Your accumulated capital after completing the project i_j will be $C_j = C_0 + \sum_{h=1}^{h=j} p_{i_h}$. The sequence must satisfy the constraint that you have sufficient capital to start the project i_{j+1} after completing the first j projects, i.e.,$C_j \geq c_{i_{j+1}}$ for each $j = 0, \ldots, k' - 1$.You want to maximize the final amount of capital, $C_{k'}$.

Solution

(High-level description)

Greedy.

(Pseudo Code)

(Correctness)

(Time complexity)

$O(n \log n)$

Problem 3: Minimum Cost Sum

Problem Description

You are given a sequence $a_1, a_2, \ldots, a_n$ of nonnegative integers, where $n \geq 1$. You are allowed to take any two numbers and add them to produce their sum. However, each such addition has a cost which is equal to the sum. The goal is to the find the sum of all the numbers in the sequence with minimum total cost. Describe an algorithm for finding the sum of the numbers in the sequence with minimum total cost. Argue the correctness of your algorithm.

Solution

(High-level description)

Greedy.
Huffman encoding.

(Pseudo Code)

(Correctness)

(Time complexity)

$O(n \log n)$

Problem 4: Speech recognition

Problem Description

We can use dynamic programming on a directed graph $G = (V, E)$ for speech recognition. Each edge $(u, v) \in E$ is labeled with a sound $\sigma(u, v)$ from a finite set Σ of sounds. The labeled graph is a formal model of a person speaking a restricted language. Each path in the graph starting from a distinguished vertex $v_0 \in V$ corresponds to a possible sequence of sounds produced by the model. The label of a directed path is defined to be the concatenation of the labels of the edges on that path.

Describe an efficient algorithm that, given an edge-labeled graph G with distinguished vertex v_0 and a sequence $s = (\sigma_1, \ldots, \sigma_k)$ of characters from Σ, returns a path in G that begins at v_0 and has s as its label, if any such path exists. Otherwise, the algorithm should return NO-SUCH-PATH. Analyze the running time of the algorithm. Clearly write any dynamic programming formulation you may use to solve this problem.

Solution

(High-level description)

(Pseudo Code)

(Correctness)

(Time complexity)

$O(k|V|^2)$