CSCE 411: Design and Analysis of Algorithms

Lecture 3: Dynamic Programming: First Week

Date: January 23, 2025 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Read chapter 15 over the course of the next few lectures (focus: 15.1, 15.2, 15.3).
- Homework 1 and intro video due this Friday (tomorrow)

Intro Poll

Question 1. The Fibonacci sequence is given by:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \text{ for } n > 1$$

What is F_8?

A 6

B 8

C 13

D 21

E 377

1 Dynamic Programming

Dynamic Programming breaks down a larger problem into smaller subproblems, similar to _____

This paradigm is often used specifically for _____ problems.

Dynamic Programming is designed for problems that satisfy two properties:

- **Optimal substructure**:

- **Overlapping subproblems**:

Question 2. Which property is also satisfied by the Divide-and-Conquer approach?

A Optimal substructure

B Overlapping subproblems

C Both of the above

D None of the above

Class activity: Fibonacci sequence The Fibonacci sequence is given by:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \text{ for } n > 1$$

Here's a naive algorithm that works for computing F_k.

FIB(n)
  if $n == 0$ **then**
    Return 0
  else if $n == 1$ **then**
    Return 1
  else
    Return FIB$(n-1)$ + FIB$(n-2)$
  end if

Some questions

1. How does this problem satisfy the two conditions for dynamic programming?
2. What's wrong or inefficient about the Fibonacci sequence? (Try using it to compute F_4 or F_5.)
3. What's a strategy for fixing the inefficiency?

2 The Rod Cutting Problem

Let p_i equal the _____

Problem 1. (The Rod Cutting Problem.) Given a rod of length n inches, determine how to cut it into pieces to:

All subpieces have an integer length in inches.

Try a few small values of i We can set up a price table, and let r_i = optimal cost for a rod of size i.

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	30
r_i										

Generalizing our strategy

Theorem 2.1. *The problem satisfies the following structure:*

Meaning: the optimal solution for n is the same as finding the best way to split it into two pieces, and then split these sub-rods in their optimal ways.

Why?

Theorem 2.2. *The problem satisfies the following even simpler characterization:*

Why?

2.1 Runtime Analysis for a Bottom-Up Approach

$$r_0 = 0$$

$$r_1 = p_1$$

$$r_2 = \max\{p_1 + r_1, p_2 + r_0\}$$

$$r_3 = \max\{p_1 + r_2, p_2 + r_1, p_3 + r_0\}$$

$$\vdots$$

$$r_n = \max\{p_1 + r_{n-1}, p_2 + r_{n-2}, \ldots, p_n + r_0\}$$

3 The Longest Common Subsequence Problem

Preliminaries A sequence is an ordered list of elements:

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$

A subsequence of X is a sequence you obtain by deleting elements of X.

A sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a subsequence of X if there are indices $i_1 < i_2 < \cdots < i_k$ such that:

$$z_1 = x_{i_1}, z_2 = x_{i_2}, \ldots, z_k = x_{i_k}.$$

Example

3.1 Defining the LCS Problem

Given two sequences

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$
$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

A *longest common subsequence* (LCS) of (X, Y) is a

Brute Force Algorithm: Consider every subsequence of X and check if it's a subsequence of Y. Return the longest sequence of X where the answer is "yes".

Question 3. How many subsequences of X are there?

A $\Theta(m)$

B $\Theta(m^2)$

C $\Theta(2^m)$

Question 4. How long does it take to check whether a sequence Z of length $k < n$ is a subsequence of $Y = \langle y_1, y_2, \ldots, y_n \rangle$

A $\Theta(1)$

B $\Theta(k)$

C $\Theta(n)$

D $\Theta(nk)$

E $\Theta(2^n)$

3.2 LCS subproblems

Given an instance of LCS (X, Y) with sequences

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$
$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

Define

$$X_i = \langle x_1, x_2, \ldots, x_i \rangle$$
$$Y_j = \langle y_1, y_2, \ldots, y_j \rangle$$

The subproblems of (X, Y) we will consider are of the form:

Question 5. How many different subproblems of this form are there?

A $\Theta(n + m)$

B $\Theta(nm)$

C $\Theta(2^n + 2^m)$

D $\Theta(2^{n+m})$

3.3 Proving optimal substructure

Theorem 3.1. *Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be an LCS of (X, Y).*

1. *If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of (X_{m-1}, Y_{n-1})*
2. *If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of (X_{m-1}, Y)*
3. *If $x_m \neq y_n$ and $z_k \neq y_n$, then Z is an LCS of (X, Y_{n-1}).*

This means: *if $x_m \neq y_n$, the LCS of (X, Y) is contained within (X, Y_{n-1})* or (X_{m-1}, Y).

Proof. See textbook. You will prove a small part of it for homework.
Illustration.

3.4 Defining a recursive approach

The theorem gives the following strategy for recursively finding the LCS of (X, Y)

- If $x_m = y_n$,

- If $x_m \neq y_n$,

Recurrence relations

$$c[i, j] = \text{ the length of an LCS of } (X_i, Y_j)$$

This satisfies the following recurrence:

Are subproblems overlapping?

4 Runtime Analysis

Recall that we have

$$c[i,j] = \text{ the length of an LCS of } (X_i, Y_j)$$

This satisfies the following recurrence:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

Question 6. *Assuming we carefully solve subproblems only once, what should be the overall runtime of finding the LCS of (X, Y) using dynamic programming? Hint: how many different subproblems are there?*

A $\Theta(mn)$

B $\Theta(mn^2)$

C $\Theta(m^2n)$

D $\Theta(m^2n^2)$

4.1 Presenting an Actual Algorithm

Think of the $c[i, j]$ values as being entries of a table or matrix.

LCS-LENGTH(X, Y)
  $m = \text{length}(X)$
  $n = \text{length}(Y)$
  Let $c[0..m, 0..n]$ be a new table
  for $i = 1$ to m **do**
    $c[i, 0] = 0$
  end for
  for $j = 1$ to n **do**
    $c[0, j] = 0$
  end for
  for $i = 1$ to m **do**
    for $j = 1$ to n **do**
      if $x_i == y_j$ **then**

      else

      end if
    end for
  end for
  Return $c[m, n]$

4.2 Finding the longest common subsequence

The algorithm on the previous page gives the length of the longest possible subsequence, but not the subsequence itself. We can obtain the subsequence by inspecting the entries of c, starting with $c[m, n]$.

The textbook solution suggests storing arrows to tell you how to build up the subsequence.