CS 362, Lecture 8

Jared Saia
University of New Mexico

Today's Outline

- Longest Common Subsequence
- Intro to Greedy Algs

Subsequence Definition

- Assume given sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$
- Let $Z = \langle z_1, z_2, \ldots, z_l \rangle$
- Then Z is a *subsequence* of X if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices such that for all $j = 1, 2, \ldots, k$, $x_{i_j} = z_j$

Example

- Let $X = \langle A, B, C, B, A, B, D, C \rangle$,
- $Z = \langle A, C, A, B, C \rangle$
- Then, Z is a subsequence of X

Common Subsequence

- Given two sequences X and Y, we say that Z is a common subsequence of X and Y if Z is a subsequence of X and Z is a subsequence of Y
- Example: $X = \langle A, B, D, C, B, A, B, C \rangle$, $Y = \langle A, D, B, C, D, B, A, B \rangle$
- Then $Z = \langle A, B, B, A, B \rangle$ is a common subsequence
- Z is not a longest common subsequence(LCS) of X and Y though since the common subsequence $Z' = \langle A, B, C, B, A, B \rangle$ is longer
- Q: Is Z' a longest common subsequence?

LCS Problem

- We are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$
- Goal: Find a maximum-length common subsequence of X and Y

Brute Force

- Brute Force approach is to enumerate all possible subsequences of X, check to see if its a subsequence of Y, and then keep track of the longest common subsequence of both X and Y
- This is slow.
- Q: How many subsequences of X are there?

Terminology

- Given a sequence $X = \langle x_1, x_2, \ldots x_m \rangle$, for $i = 0, 1, \ldots, m$, let X_i be the i-th prefix of X i.e. $X_i = \langle x_1, x_2, \ldots, x_i \rangle$
- Example: if $X = \langle A, B, D, C \rangle$, $X_0 = \langle \rangle$ and $X_3 = \langle A, B, D \rangle$

Optimal Substructure

Lemma 1: Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and let $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then:

- If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is a LCS of X_{m-1} and Y_{n-1}
- If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is a LCS of X_{m-1} and Y
- If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}

In-Class Exercise

- Prove each of the three statements in the previous slide
- Hint: Use proof by contradiction

Recursive Solution

- Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be arbitrary sequences
- Based on Lemma 1, there are two main possibilities for the LCS of X and Y:
  - If $x_m = y_n$, $LCS(X, Y)$ is $LCS(X_{m-1}, Y_{n-1})$ appended to $x_m = y_n$
  - Otherwise, either $LCS(X, Y)$ is $LCS(X_{m-1}, Y)$ or $LCS(X, Y_{n-1})$ (whichever is larger)

Recursive solution

- Let $c(i, j)$ be the length of an LCS of the sequence X_i, Y_j
- Note that $c(i, j) = 0$ if i or j is 0
- Thus we have:

$$\boxed{\begin{array}{ll} c(i,j) = 0 & \text{if } i = 0 \text{ or } j = 0 \\ c(i,j) = c(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ c(i,j) = \max(c(i, j-1), c(i-1, j)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{array}}$$

DP Solution

- This is already enough to write up a recursive function, however the naive recursive function will take exponential time
- Instead, we can use dynamic programming and solve from the bottom up
- Code for doing this is on p. 353 and 355 of the text, basically it uses the same ideas we've seen before of filling in entries in a table from the bottom up.

Example

- Consider $X = \langle A, B, D, C, B, A, B, C \rangle$, $Y = \langle A, D, B, C, D, B, A, B \rangle$
- The next slide gives the table constructed by the DP algorithm for computing the LCS of X and Y
- The bold numbers represent one possible path giving a LCS.
- The arrows keep track of where the minimum is obtained from

Example

		A	B	D	C	B	A	B	C
	0	0	0	0	0	0	0	0	0
A	0	**1**→	1→	1→	1→	1→	1→	1→	1
D	0	**1**	1	2→	2→	2→	2→	2→	2
B	0	1	**2**→	**2**→	2	3→	3→	3→	3
C	0	1	2→	2	**3**→	3→	3→	3→	4
D	0	1	2	3→	3→	3→	3→	3	4
B	0	1	2	3→	3	**4**→	4→	4→	4
A	0	1	2	3→	3	4	**5**→	5→	5
B	0	1	2	3→	3	4	5	**6**→	**6**

Reconstruction

- To find a reconstruction, first find a path of edges leading from the bottom right corner to the top left corner
- In this path, the target of each diagonal arrow gives a character to include in the LCS
- In our example, $\langle A, B, C, B, A, B \rangle$ is the LCS we get by following the edges along a path from the bottom right to the top left

Take Away

- We've seen four different DP type algorithms
- In each case, we did the following 1) found a recurrence for the solution 2) built solutions to the recurrence from the bottom up
- You should be prepared to do this on your own now!

Greedy Algorithms

"Greed is Good" - Michael Douglas- in Wall Street

- A greedy algorithm always makes the choice that looks best at the moment
- Greedy algorithms do not always lead to optimal solutions, but for many problems they do
- In the next week or two, we will see several problems for which greedy algorithms produce optimal solutions including: activity selection, fractional knapsack, and making change
- When we study graph theory, we will also see that greedy algorithms work well for computing shortest paths and finding minimum spanning trees.

Activity Selection

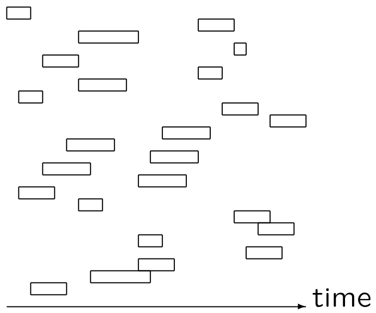
- You are given a list of programs to run on a single processor
- Each program has a start time and a finish time
- However the processor can only run one program at any given time, and there is no preemption (i.e. once a program is running, it must be completed)

Another Motivating Problem

- Suppose you are at a film fest, all movies look equally good, and you want to see as many complete movies as possible
- This problem is also exactly the same as the activity selection problem.

Example

Imagine you are given the following set of start and stop times for activities



Ideas

- There are many ways to optimally schedule these activities
- Brute Force: examine every possible subset of the activites and find the largest subset of non-overlapping activities
- Q: If there are n activities, how many subsets are there?
- The book also gives a DP solution to the problem

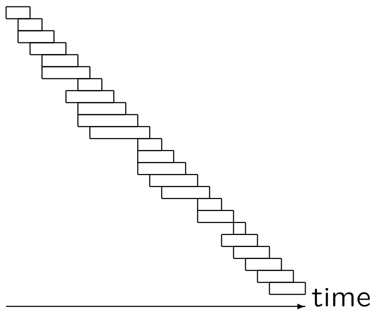
Greedy Activity Selector

1. Sort the activities by their finish times
2. Schedul the first activity in this list
3. Now go through the rest of the sorted list in order, scheduling activities whose start time is after (or the same as) the last scheduled activity

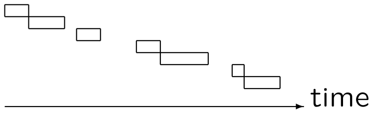
(note: code for this algorithm is in section 16.1)

Greedy Algorithm

Sorting the activities by their finish times



Greedy Scheduling of Activities



Analysis

- Let n be the total number of activities
- The algorithm first sorts the activities by finish time taking $O(n \log n)$
- Then the algorithm visits each activity exactly once, doing a constant amount of work each time. This takes $O(n)$
- Thus total time is $O(n \log n)$

Optimality

- The big question here is: Does the greedy algorithm give us an optimal solution???
- Surprisingly, the answer turns out to be yes

Todo

- Finish Chapter 15
- Start Chapter 16