CS420+500: Advanced Algorithm Design and Analysis

Lectures: Feb 27 + Mar 01, 2017

Prof. Will Evans *Scribe: Rodrigo Araujo*

In this lecture we talked about dynamic programming and some problems to motivate the use of such technique, these problems are:

1. Longest Common Subsequence (LCS);
2. Longest Increasing Subsequence (LIS);
3. Introduced the idea of developing a faster solution to LIS.

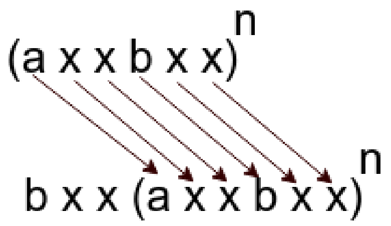
1 Dynamic Programming: Longest Common Subsequence

Problem: Longest Common Subsequence

Given two character strings: X composed with m characters, and Y composed with n characters, find the *Longest Common Subsequence* (LCS) of X and Y.

A *subsequence* of a given string X is $X[i_1] \cdot X[i_2] \cdot ... \cdot X[i_k]$ where $i_1 < i_2 < ... < i_k$. Note that the dot symbol $(\cdot)$ stands for string concatenation.

One could think of solving this problem by following a greedy approach like:



However, it's possible to see that we will achieve a very bad performance if we follow this approach.

This is an optimization problem, it has a recursive solution *if* you can reduce the problem to its smallest form, so *Dynamic Programming* might be a good approach.

Algorithm 1: LCS

```
if $X[m] = Y[n]$ then
  | return 1 + $LCS(X[1..n-1], Y[1..n-1])$
else if $X[m] \neq Y[n]$ then
  | return $max(LCS(X[1..m-1], Y[1..n]),\ LCS(X[1..m], Y[1..n-1]))$
```

How many subproblems do we have to solve? The answer is $N * M$ subproblems. They are $LCS(X[1..i], Y[1..j])$ for $1 \leq i \leq m,\ 1 \leq j \leq n$

1.1 Tabulate

Define a table $F[i,j] = \mathit{length\ \ of\ \ LCS}(X[1..i], Y[1..j])$

$$F[i,j] = \begin{cases} F[i-1,j-1], & \text{if} X[i] = Y[j] \\ max(F[i,j-1], F[i-1,j]), & X[i] \neq Y[j] \end{cases}$$

After we build the table, we can traverse it and get the actual sequence.

The runtime is $O(nm)$.

2 Longest Increasing Subsequence

Problem: Given a sequence of numbers $R[1..n]$ find the longest increasing subsequence (LIS) of R.

Example: $R = 5, 3, 4, 9, 6, 2, 1$

$LIS(R) = 3, 4, 6, 8$

We can solve this problem by using *Longest Common Subsequence*:

1. Report $LCS(R, Sort(R))$

The runtime for this approach is $O(n^2)$

2.1 Faster solution to LIS

Suppose we are trying to find the LIS of $R[1..k]$, what information about $R[1..k-1]$ would be useful? We have a few options:

1. The LIS of $R[1..k-1]$
2. Best *(as in one with smallest last value)* LIS of $R[1..k-1]$
3. Best Increasing Subsequence of all possible lengths $1, 2, 3...j = BIS[1], BIS[2], ...$

The first two options are not enough. So we use the third option, for example:

$8, 3, 4, 9, 6, 2, 1, 5, 7, 6$

After 1, before 5:

$$BIS[1] = 1$$

$$BIS[2] = 3, 4$$

$$BIS[3] = 3, 4, 6$$

$$BIS[4] = ..., one$$

After 5, before 7:

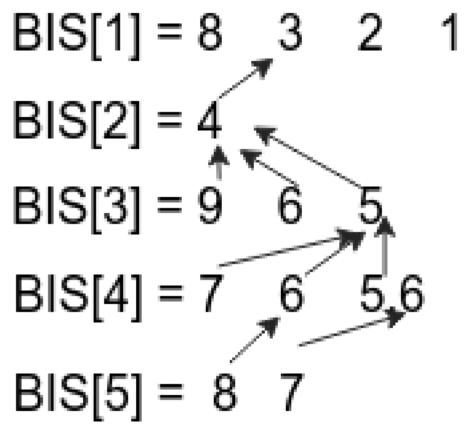
$$BIS[1] = 1$$
$$BIS[2] = 3, 4$$
$$BIS[3] = 3, 4, 5$$
$$BIS[4] = ..., one$$

After 7, before 6:

$$BIS[1] = 1$$
$$BIS[2] = 3, 4$$
$$BIS[3] = 3, 4, 5$$
$$BIS[4] = 3, 4, 5, 7$$

Let's consider another example:

$R = 8, 3, 4, 9, 6, 2, 1, 5, 7, 6, 8, 7, 5.6$



The logic behind this solution is: you should always add an arrow from the last element n to the last element of $BIS[n-1]$. $BIS[i]$ is a list of the end numbers of the best increasing subsequence of length i as these numbers are encountered in left to right sweep of the input.

To add the next number in R, perform binary search on the last values in the BISs and put the next number in sorted order (point to the end value of the previous BIS).

This way we achieve a runtime of $O(nlogn)$.