CS173
Longest Increasing Substrings

Tandy Warnow

CS 173
Longest Increasing Substrings
Tandy Warnow

Today's material

- The Longest Increasing Subsequence problem
- DP algorithm for finding a longest increasing substring

Dynamic Programming

Dynamic programming is an algorithmic design technique that can make it easy to solve problems efficiently.

Dynamic programming is similar to recursion – but it is bottom-up, instead of top-down.

Interesting applications of dynamic programming include:

- Computing the longest increasing subsequence in a sequence
- Finding the longest common subsequence of two sequences
- Finding all-pairs shortest paths in an edge-weighted graph
- Solving two-player games

Finding a Longest Increasing Subsequence

Input: sequence $X = x_1, x_2, \ldots, x_n$ of integers
Output: longest subsequence of X that is strictly increasing

Example: $X = 7, 1, 4, 3, 5, 2, 4, -1, 6, 1, 2, 5, 6, 7$

Some increasing subsequences:

- 3,5
- -1,2,5
- 1,3,5,6,7
- -1,1,2,5,6,7

Maybe the last one is the longest?

Finding the longest increasing subsequence in a sequence can be done in polynomial time using dynamic programming.

We will solve the *simpler* problem of finding the longest increasing substring.

Finding a Longest Increasing Substring

Input: sequence (or array) $X = x_1 x_2 \ldots x_n$ of integers
Output: increasing substring of X that is as long as possible.

What is a substring?

- A substring is a string that begins at some x_i and ends at some x_j (with $j \geq i$) and includes all the intermediate elements.

For example, x_2, x_3, x_4 is a substring but x_2, x_4 is not.

Suppose $X = 1, 3, 1, 8, 2, 4, 9, 2, 10, 3$.

- What are some increasing substrings?
- What are some increasing subsequences?

Finding a Longest Increasing Substring

Input: sequence (or array) $X = x_1 x_2 \ldots x_n$ of integers
Output: increasing substring of X that is as long as possible.

Example: $X = 1, 3, 1, 8, 2, 4, 9, 2, 10, 3$ (so $x_1 = 1, x_2 = 3$, etc.)

Which of the following are increasing substrings?

1. x_1
2. x_5
3. x_1, x_3
4. x_1, x_2
5. x_2, x_3
6. x_3, x_4

Finding a Longest Increasing Substring

Example: $X = 1, 3, 1, 8, 2, 4, 9, 2, 10, 3$ (so $x_1 = 1, x_2 = 3$, etc.)

By inspection we see that the longest increasing substring is $2, 4, 9$, formed by using x_5, x_6, x_7.

How can we design an algorithm to solve this problem?

Finding a Longest Increasing Substring

Example: $X = 1, 3, 1, 8, 2, 4, 9, 2, 10, 3$ (so $x_1 = 1, x_2 = 3$, etc.)

How can we design an algorithm to solve this problem?

Let $M[i]$ denote the length of the longest increasing substring that ends at x_i.

So:

- $M[1] = 1$
- $M[2] = 2$
- $M[3] = 1$
- $M[4] = 2$ (why isn't it 3?)

Class exercise:

1. calculate $M[i]$ for $i = 5, 6, 7, 8, 9, 10$.
2. What is the longest increasing substring for X?
3. What index does it end at?
4. What do you see for $M[i]$ for that index i?

Finding a Longest Increasing Substring

Let $M[i]$ denote the length of the longest increasing substring that ends at x_i.

Suppose X is your arbitrary input.

How can we answer these two questions:

1. If we knew $M[1], M[2], \ldots, M[n]$ (where n is the length of the array), what would be the length of the longest incrasing substring for X? Would it be $M[n]$ or something else?
2. Can we use $M[1], M[2], \ldots, M[j-1]$ to compute $M[j]$?

Computing $M[i]$

Let $M[i]$ denote the length of the longest increase substring that ends at x_i.

Then how we set $M[i]$ depends on the value of i:

1. If $i = 1$ then $M[i] = 1$
2. If $i \geq 2$ then:
    - $M[i] = 1$ if $x_{i-1} \geq x_i$
    - $M[i] = 1 + M[i-1]$ if $x_{i-1} < x_i$

Why is this correct?

Computing $M[i]$

Let $M[i]$ denote the length of the longest increase substring that ends at x_i.
Then:

1. $M[1] = 1$.
   - Because x_1 is the longest increasing substring that ends at x_1
2. $M[i] = 1$ if $x_{i-1} \geq x_i$ and $i \geq 2$
   - Because x_i is the longest increasing substring ending at x_i when $x_{i-1} \geq x_i$
3. $M[i] = 1 + M[i-1]$ if $x_{i-1} < x_i$ and $i \geq 2$
   - Because the longest increasing substring ending at x_i in this case is formed by appending x_i to the longest increasing substring ending at x_{i-1}

Putting this together

Given $X = x_1, x_2, \ldots, x_n$, to find the *length* of the longest increasing substring:

- For $i = 1$ up to n do:
  - Compute $M[i]$ using rules from previous slide
- Return $\max\{M[1], M[2], M[3], \ldots, M[n]\}$

Questions:

1. Why is this correct?
2. What is the running time?
3. This only gives you the length of the longest increasing substring.
4. How do you get the longest increasing substring itself?

Writing DP algorithms

Please observe the following guidelines for writing a dynamic programming algorithm:

- Explain your variables using English, showing what they are supposed to mean
- Show how to compute the values for the boundary conditions
- Specify the order in which you compute the values
- Show how to compute each value based on the earlier computations
- Show where the final answer is stored

Summary

- Dynamic programming and recursive algorithms are two ways of dealing with algorithm design.
- One is top down (recursion) and the other is bottom-up (dynamic programming).
- You can **prove** your algorithm is correct using induction, when the algorithm uses recursion or dynamic programming.

In both cases, you identify subproblems and show how solving subproblems lets you solve big problems.