Longest Increasing Subsequence

2013 IOI Camp 1

Robert Spencer

December 11, 2013

Introduction

Given some sequence, what is the longest, increasing subsequence?

Introduction

Given some sequence, what is the longest, increasing subsequence?

0 8 4 14 2 10 6 12 1 16

Introduction

Given some sequence, what is the longest, increasing subsequence?

0	**8**	4	14	2	**10**	6	**12**	1	**16**

But how to do that?

First Attempt

How about trying all different subsequences?

First Attempt

How about trying all different subsequences?
Guarranteed to get correct solution. Yay!

First Attempt

How about trying all different subsequences?
Guarranteed to get correct solution. Yay!
But what is its efficiency?

First Attempt

How about trying all different subsequences?
Guarranteed to get correct solution. Yay!
But what is its efficiency?
$O(2^n)$! Yikes!

Second Attempt

So how do we better this? From the constraints, we know that the solution has to be fairly fast. Also, we can intuit that we can "build" on smaller subproblems to get the solution to the big problem.

Second Attempt

So how do we better this? From the constraints, we know that the solution has to be fairly fast. Also, we can intuit that we can “build” on smaller subproblems to get the solution to the big problem.
Also, you have just had a lecture on DP.

Second Attempt

So how do we better this? From the constraints, we know that the solution has to be fairly fast. Also, we can intuit that we can "build" on smaller subproblems to get the solution to the big problem.
Also, you have just had a lecture on DP.
What is the semi-obvious DP?

Second Attempt

So how do we better this? From the constraints, we know that the solution has to be fairly fast. Also, we can intuit that we can "build" on smaller subproblems to get the solution to the big problem.
Also, you have just had a lecture on DP.
What is the semi-obvious DP?
Store the length of the longest increasing subsequence ending on that point. This can also be used to reconstruct the subsequence.

0	8	4	14	2	10	6	12	1	16

Second Attempt

So how do we better this? From the constraints, we know that the solution has to be fairly fast. Also, we can intuit that we can "build" on smaller subproblems to get the solution to the big problem.
Also, you have just had a lecture on DP.
What is the semi-obvious DP?
Store the length of the longest increasing subsequence ending on that point. This can also be used to reconstruct the subsequence.

0	8	4	14	2	10	6	12	1	16
1	2	2	3	2	3	4	5	2	6

Doing the DP

How do we find dp[i] though?

Doing the DP

How do we find dp[i] though?

```
for i from 1 to n do
  best = 0
  for j from 1 to i-1 do
    if s[j] < s[i] and dp[j] > best then
      best = dp[j]
  dp[i] = best + 1
```

Doing the DP

How do we find dp[i] though?

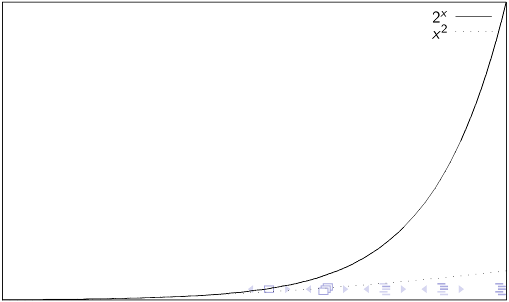
```
for i from 1 to n do
  best = 0
  for j from 1 to i-1 do
    if s[j] < s[i] and dp[j] > best then
      best = dp[j]
  dp[i] = best + 1
```

Efficiency $O(n^2)$ is "somewhat better."

How do we find dp[i] though?

```
for i from 1 to n do
  best = 0
  for j from 1 to i-1 do
    if s[j] < s[i] and dp[j] > best then
      best = dp[j]
  dp[i] = best + 1
```

Efficiency $O(n^2)$ is "somewhat better."



Improvements

We can do better though...

Improvements

We can do better though...
The inner loop is the problematic one. It adds a factor of n to our $O(n^2)$.

We can do better though...
The inner loop is the problematic one. It adds a factor of n to our $O(n^2)$.
We also don't have a very easy way of finding the sequence (one exists but it can be bettered)

Improvements

We can do better though...

The inner loop is the problematic one. It adds a factor of n to our $O(n^2)$.

We also don't have a very easy way of finding the sequence (one exists but it can be bettered)

A different DP is needed.

Third Attempt

If we have a choice amongst previous elements when building our LIS, we might as well take the smallest. This leads to the DP:

- Let $m[j]$ store the position k of the smallest $s[k]$ such that there is a increasing subsequence of length j ending on $s[k]$.
- Let $p[i]$ store the predecessor of $s[i]$ in the longest increasing subsequence ending on $s[i]$.

Third Attempt

If we have a choice amongst previous elements when building our LIS, we might as well take the smallest. This leads to the DP:

- Let $m[j]$ store the position k of the smallest $s[k]$ such that there is a increasing subsequence of length j ending on $s[k]$.
- Let $p[i]$ store the predecessor of $s[i]$ in the longest increasing subsequence ending on $s[i]$.

It is important to note that $s[m[1]], s[m[2]], \ldots, s[m[L]]$ is nondecreasing. This is true, as if there is a increasing subsequence of length i ending at $s[m[i]]$, then there is also a increasing subsequence of length $i - 1$ ending at a smaller value, i.e. the all-but-one of that sequence.

Third Attempt

If we have a choice amongst previous elements when building our LIS, we might as well take the smallest. This leads to the DP:

- Let $m[j]$ store the position k of the smallest $s[k]$ such that there is a increasing subsequence of length j ending on $s[k]$.
- Let $p[i]$ store the predecessor of $s[i]$ in the longest increasing subsequence ending on $s[i]$.

It is important to note that $s[m[1]], s[m[2]], \ldots, s[m[L]]$ is nondecreasing. This is true, as if there is a increasing subsequence of length i ending at $s[m[i]]$, then there is also a increasing subsequence of length $i - 1$ ending at a smaller value, i.e. the all-but-one of that sequence.
Then we can build this up as follows:

Psudocode

```
L = 0
for i = 1 to n do
  binary search for the largest positive j  L
    such that s[m[j]] < s[i] (or set j = 0 if no such value
  P[i] = m[j]
  if j == L or s[i] < s[m[j+1]]:
    m[j+1] = i
    L = max(L, j+1)
```

Psudocode

```
L = 0
for i = 1 to n do
  binary search for the largest positive j  L
    such that s[m[j]] < s[i] (or set j = 0 if no such value
  P[i] = m[j]
  if j == L or s[i] < s[m[j+1]]:
    m[j+1] = i
    L = max(L, j+1)
```

This has $O(n \log n)$ which is good enough for most cases.