13.4

Longest Increasing Subsequence Revisited

13.4.1

Longest Increasing Subsequence

Sequences

Definition 13.1.

Sequence: an ordered list $a_1, a_2, \ldots, a_n$. **Length** of a sequence is number of elements in the list.

Definition 13.2.

$a_{i_1}, \ldots, a_{i_k}$ is a **subsequence** of $a_1, \ldots, a_n$ if $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

Definition 13.3.

A sequence is **increasing** if $a_1 < a_2 < \ldots < a_n$. It is **non-decreasing** if $a_1 \leq a_2 \leq \ldots \leq a_n$. Similarly **decreasing** and **non-increasing**.

Sequences

Example...

Example 13.4.

1. Sequence: $6, 3, 5, 2, 7, 8, 1, 9$
2. Subsequence of above sequence: $5, 2, 1$
3. Increasing sequence: $3, 5, 9, 17, 54$
4. Decreasing sequence: $34, 21, 7, 5, 1$
5. Increasing subsequence of the first sequence: $2, 7, 9$.

Longest Increasing Subsequence Problem

Input A sequence of numbers $a_1, a_2, \ldots, a_n$

Goal Find an **increasing subsequence** $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of maximum length

Example 13.5.

1. Sequence: 6, 3, 5, 2, 7, 8, 1
2. Increasing subsequences: 6, 7, 8 and 3, 5, 7, 8 and 2, 7 etc
3. Longest increasing subsequence: 3, 5, 7, 8

Longest Increasing Subsequence Problem

Input A sequence of numbers $a_1, a_2, \ldots, a_n$

Goal Find an **increasing subsequence** $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of maximum length

Example 13.5.

1. Sequence: 6, 3, 5, 2, 7, 8, 1
2. Increasing subsequences: 6, 7, 8 and 3, 5, 7, 8 and 2, 7 etc
3. Longest increasing subsequence: 3, 5, 7, 8

Recursive Approach: Take 1

LIS: Longest increasing subsequence

Can we find a recursive algorithm for LIS?

LIS($A[1..n]$):

1. Case 1: Does not contain $A[n]$ in which case $\text{LIS}(A[1..n]) = \text{LIS}(A[1..(n-1)])$
2. Case 2: contains $A[n]$ in which case $\text{LIS}(A[1..n])$ is not so clear.

Observation 13.6.

For second case we want to find a subsequence in $A[1..(n-1)]$ that is restricted to numbers less than $A[n]$. This suggests that a more general problem is **LIS_smaller**$(A[1..n], x)$ *which gives the longest increasing subsequence in A where each number in the sequence is less than x.*

Recursive Approach: Take 1

LIS: Longest increasing subsequence

Can we find a recursive algorithm for LIS?

LIS($A[1..n]$):

1. Case 1: Does not contain $A[n]$ in which case $\mathrm{LIS}(A[1..n]) = \mathrm{LIS}(A[1..(n-1)])$
2. Case 2: contains $A[n]$ in which case $\mathrm{LIS}(A[1..n])$ is not so clear.

Observation 13.6.

For second case we want to find a subsequence in $A[1..(n-1)]$ that is restricted to numbers less than $A[n]$. This suggests that a more general problem is **LIS_smaller**$(A[1..n], x)$ *which gives the longest increasing subsequence in A where each number in the sequence is less than x.*

Recursive Approach

$LIS(A[1..n])$: the length of longest increasing subsequence in A

LIS_smaller$(A[1..n], x)$: length of longest increasing subsequence in $A[1..n]$ with all numbers in subsequence less than x

```
LIS_smaller(A[1..i], x):
    if $i = 0$ then return 0
    $m = $ LIS_smaller$(A[1..i-1], x)$
    if $A[i] < x$ then
        $m = max(m, 1 + $ LIS_smaller$(A[1..i-1], A[i]))$
    Output $m$
```

```
LIS(A[1..n]):
    return LIS_smaller$(A[1..n], \infty)$
```

Recursive Approach

```
LIS_smaller($A[1..i], x$):
    if $i = 0$ then return 0
    $m$ = LIS_smaller($A[1..i-1], x$)
    if $A[i] < x$ then
        $m = max(m, 1 + \text{LIS\_smaller}(A[1..i-1], A[i]))$
    Output $m$
```

```
LIS($A[1..n]$):
    return LIS_smaller($A[1..n], \infty$)
```

- How many distinct sub-problems will **LIS_smaller**$(A[1..n], \infty)$ generate? $O(n^2)$
- What is the running time if we memoize recursion? $O(n^2)$ since each call takes $O(1)$ time to assemble the answers from to recursive calls and no other computation.
- How much space for memoization? $O(n^2)$

Recursive Approach

```
LIS_smaller($A[1..i], x$):
    if $i = 0$ then return 0
    $m$ = LIS_smaller($A[1..i-1], x$)
    if $A[i] < x$ then
        $m = max(m, 1 + \text{LIS\_smaller}(A[1..i-1], A[i]))$
    Output $m$
```

```
LIS($A[1..n]$):
    return LIS_smaller($A[1..n], \infty$)
```

- How many distinct sub-problems will **LIS_smaller**$(A[1..n], \infty)$ generate? $O(n^2)$
- What is the running time if we memoize recursion? $O(n^2)$ since each call takes $O(1)$ time to assemble the answers from to recursive calls and no other computation.
- How much space for memoization? $O(n^2)$

Recursive Approach

```
LIS_smaller($A[1..i], x$):
    if $i = 0$ then return 0
    $m$ = LIS_smaller($A[1..i-1], x$)
    if $A[i] < x$ then
        $m = max(m, 1 +$ LIS_smaller($A[1..i-1], A[i]$))
    Output $m$
```

```
LIS($A[1..n]$):
    return LIS_smaller($A[1..n], \infty$)
```

- How many distinct sub-problems will **LIS_smaller**$(A[1..n], \infty)$ generate? $O(n^2)$
- What is the running time if we memoize recursion? $O(n^2)$ since each call takes $O(1)$ time to assemble the answers from to recursive calls and no other computation.
- How much space for memoization? $O(n^2)$

Recursive Approach

```
LIS_smaller($A[1..i], x$):
    if $i = 0$ then return 0
    $m$ = LIS_smaller($A[1..i-1], x$)
    if $A[i] < x$ then
        $m = max(m, 1 +$ LIS_smaller($A[1..i-1], A[i]$))
    Output $m$
```

```
LIS($A[1..n]$):
    return LIS_smaller($A[1..n], \infty$)
```

- How many distinct sub-problems will **LIS_smaller**$(A[1..n], \infty)$ generate? $O(n^2)$
- What is the running time if we memoize recursion? $O(n^2)$ since each call takes $O(1)$ time to assemble the answers from to recursive calls and no other computation.
- How much space for memoization? $O(n^2)$

Recursive Approach

```
LIS_smaller($A[1..i], x$):
    if $i = 0$ then return 0
    $m$ = LIS_smaller($A[1..i-1], x$)
    if $A[i] < x$ then
        $m = max(m, 1 + \text{LIS\_smaller}(A[1..i-1], A[i]))$
    Output $m$
```

```
LIS($A[1..n]$):
    return LIS_smaller($A[1..n], \infty$)
```

- How many distinct sub-problems will **LIS_smaller**$(A[1..n], \infty)$ generate? $O(n^2)$
- What is the running time if we memoize recursion? $O(n^2)$ since each call takes $O(1)$ time to assemble the answers from to recursive calls and no other computation.
- How much space for memoization? $O(n^2)$

Recursive Approach

```
LIS_smaller($A[1..i], x$):
    if $i = 0$ then return $0$
    $m$ = LIS_smaller($A[1..i-1], x$)
    if $A[i] < x$ then
        $m = max(m, 1 +$ LIS_smaller($A[1..i-1], A[i]$))
    Output $m$
```

```
LIS($A[1..n]$):
    return LIS_smaller($A[1..n], \infty$)
```

- How many distinct sub-problems will **LIS_smaller**($A[1..n], \infty$) generate? $O(n^2)$
- What is the running time if we memoize recursion? $O(n^2)$ since each call takes $O(1)$ time to assemble the answers from to recursive calls and no other computation.
- How much space for memoization? $O(n^2)$

Naming subproblems and recursive equation

After seeing that number of subproblems is $O(n^2)$ we name them to help us understand the structure better. For notational ease we add ∞ at end of array (in position $n+1$)

$\mathrm{LIS}(i, j)$: length of longest increasing sequence in $A[1..i]$ among numbers less than $A[j]$ (defined only for $i < j$)

Base case: $\mathrm{LIS}(0, j) = 0$ for $1 \leq j \leq n+1$

Recursive relation:

- $\mathrm{LIS}(i, j) = \mathrm{LIS}(i-1, j)$ if $A[i] > A[j]$
- $\mathrm{LIS}(i, j) = \max\{LIS(i-1, j), 1 + LIS(i-1, i)\}$ if $A[i] \leq A[j]$

Output: $\mathrm{LIS}(n, n+1)$.

Naming subproblems and recursive equation

After seeing that number of subproblems is $O(n^2)$ we name them to help us understand the structure better. For notational ease we add ∞ at end of array (in position $n+1$)

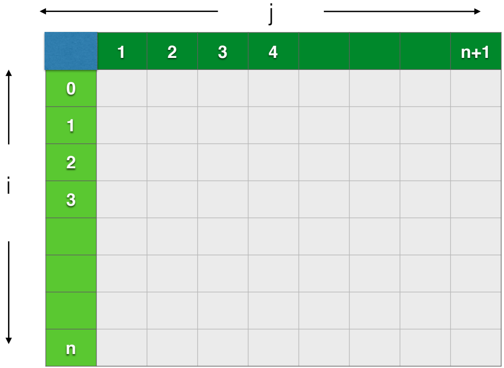
$\mathrm{LIS}(i,j)$: length of longest increasing sequence in $A[1..i]$ among numbers less than $A[j]$ (defined only for $i < j$)

Base case: $\mathrm{LIS}(0,j) = 0$ for $1 \leq j \leq n+1$
Recursive relation:

- $\mathrm{LIS}(i,j) = \mathrm{LIS}(i-1,j)$ if $A[i] > A[j]$
- $\mathrm{LIS}(i,j) = \max\{LIS(i-1,j), 1 + LIS(i-1,i)\}$ if $A[i] \leq A[j]$

Output: $\mathrm{LIS}(n,n+1)$.

How to order bottom up computation?



	1	2	3	4				n+1
0								
1								
2								
3								
n								

Sequence: $A[1..7] = 6, 3, 5, 2, 7, 8, 1$

Recursive relation:

$$\mathrm{LIS}(i, j) = \begin{cases} 0 & i = 0 \\ \mathrm{LIS}(i-1, j) & A[i] > A[j] \\ \max \begin{cases} \mathrm{LIS}(i-1, j) \\ 1 + \mathrm{LIS}(i-1, i) \end{cases} & A[i] \leq A[j] \end{cases}$$

Iterative algorithm

The dynamic program for longest increasing subsequence

```
LIS-Iterative($A[1..n]$):
    $A[n+1] = \infty$
    int $LIS[0..n, 1..n+1]$
    for $j = 1 \dots n+1$) do $LIS[0, j] = 0$

    for $i = 1 \dots n$) do
        for ($j = i + 1 \dots n$ do
            if ($A[i] > A[j]$)
                $LIS[i, j] = LIS[i-1, j]$
            else
                $LIS[i, j] = \max(LIS[i-1, j], 1 + LIS[i-1, i])$

    Return $LIS[n, n+1]$
```

Running time: $O(n^2)$
Space: $O(n^2)$

Two comments

Question: Can we compute an optimum solution and not just its value?
Yes! See notes.

Question: Is there a faster algorithm for LIS? Yes! Using a different recursion and optimizing one can obtain an $O(n \log n)$ time and $O(n)$ space algorithm. $O(n \log n)$ time is not obvious. Depends on improving time by using data structures on top of dynamic programming.

Two comments

Question: Can we compute an optimum solution and not just its value?
Yes! See notes.

Question: Is there a faster algorithm for LIS? Yes! Using a different recursion and optimizing one can obtain an $O(n \log n)$ time and $O(n)$ space algorithm. $O(n \log n)$ time is not obvious. Depends on improving time by using data structures on top of dynamic programming.

Two comments

Question: Can we compute an optimum solution and not just its value?
Yes! See notes.

Question: Is there a faster algorithm for LIS? Yes! Using a different recursion and optimizing one can obtain an $O(n \log n)$ time and $O(n)$ space algorithm. $O(n \log n)$ time is not obvious. Depends on improving time by using data structures on top of dynamic programming.

THE END

…

(for now)