Even More DP

CSE 421 Fall 22
Lecture 14

Maximum Contiguous Subarray Sum

We saw an $O(n \log n)$ divide and conquer algorithm.

Can we do better with DP?

Given: Array $A[]$

Output: i, j such that $A[i] + A[i+1] + \cdots + A[j]$ is maximized.

Dynamic Programming Process

1. Define the object you’re looking for

2. Write a recurrence to say how to find it

3. Design a memoization structure

4. Write an iterative algorithm

Maximum Contiguous Subarray Sum

We saw an $O(n \log n)$ divide and conquer algorithm.

Can we do better with DP?

Given: Array $A[]$

Output: i, j such that $A[i] + A[i+1] + \cdots + A[j]$ is maximized.

For today: just output the value$A[i] + A[i+1] + \cdots + A[j]$.

State what you want OPT(i) to be in English, is that enough to do the recursion?

Trying to Recurse

0	1	2	3	4	5	6	7
5	-6	3	4	-5	2	2	4

$OPT(3)$ would give i =2, $j = 3$

$OPT(4)$ would give $i = 2, j = 3$ too

$OPT(7)$ would give $i = 2, j = 7$ – we need to suddenly backfill with a bunch of elements that weren't optimal…

How do we make a decision on index 7? What information do we need?

What do we need for recursion?

If index i IS going to be included

We need the best subarray **that includes index $\boldsymbol{i-1}$**

If we include anything to the left, we'll definitely include index $i-1$ (because of the contiguous requirement)

If index i isn't included

We need the best subarray up to $i-1$, regardless of whether $i-1$ is included.

Two Values

Pollev.com/robbie

Need two recursive values:

$INCLUDE(i)$: sum of the maximum sum subarray among elements from 0 to i **that includes index $\boldsymbol{i}$** in the sum

$OPT(i)$: sum of the maximum sum subarray among elements 0 to i (that might or might not include i)

How can you calculate these values? Try to write recurrence(s), then think about memoization and running time.

Recurrences

$$INCLUDE(i) = \begin{cases} \max\{A[i], A[i] + INLCUDE(i-1)\} & \text{if } i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$OPT(i) = \begin{cases} \max\{INCLUDE(i), OPT(i-1)\} & \text{if } i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

If we include i, the subarray must be either just i or also include $i-1$.

Overall, we might or might not include i. If we don't include i, we only have access to elements $i-1$ and before. If we do, we want $INCLUDE(i)$ by definition.

Example

	0	1	2	3	4	5	6	7
A	5	-6	3	4	-5	2	2	4

	0	1	2	3	4	5	6	7
$OPT(i)$	5							

	0	1	2	3	4	5	6	7
$INCLUDE(i)$	5							

Example

A

0	1	2	3	4	5	6	7
5	-6	3	4	-5	2	2	4

$OPT(i)$

0	1	2	3	4	5	6	7
5	5						

$INCLUDE(i)$

0	1	2	3	4	5	6	7
5	-1						

Example

	0	1	2	3	4	5	6	7
A	5	-6	3	4	-5	2	2	4

	0	1	2	3	4	5	6	7
$OPT(i)$	5	5	5	7	7	7	7	10

	0	1	2	3	4	5	6	7
$INCLUDE(i)$	5	-1	3	7	2	4	6	10

Pseudocode

```
int maxSubarraySum(int[] A)
  int n=A.length
  int[] OPT = new int[n]
  int[] Inc = new int[n]
  inc[0]=A[0]; OPT[0] = max{A[0],0}
  for(int i=0;i<n;i++)
    inc[i]=max{A[i], A[i]+inc[i-1]}
    OPT[i]=max{inc[i], opt[i-1]}
  endFor
return OPT[n-1]
```

Recursive Thinking In General

As before, the hardest part is designing the recurrence.

It sometimes helps to think from multiple different angles.

Top-down: What's the first step to take?

Baby Yoda will first go left or down. Use recursion to find out which of left or down is better.

The farthest right operation in the string transformation will be one of insert, delete, substitute, match for free. Use recursion to find out which is best.

Recursive Thinking In General

Bottom-Up: What information could a recursive call give me that would help?

How does a path through most of the map help Baby Yoda?

Well we just need to know the values one left and one down.

The edit distance between which strings would help us compute the edit distance between our strings?

Well if we know the distance between $x_1 \dots x_{i-1}$ and $y_1 \dots y_{j-1}$ then that would tell us what happens if we substitute…that might lead you to insertions and deletions too.

Recursive Thinking In General

Some people refer to the “Optimal Substructure Property”

From the optimum (most eggs, fewest number of string operations) for a slightly smaller problem (Baby Yoda starting closer to the end, slightly smaller strings), we need to be able to build up the optimum for the full problem.

Longest Increasing Subsequence

0	1	2	3	4	5	6	7
5	-6	3	6	-5	2	8	10

Longest set of (not necessarily consecutive) elements that are increasing

5 is optimal for the array above

(indices $1,2,3,6,7$; elements $-6,3,6,8,10$)

For simplicity – assume all array elements are distinct.

Longest Increasing Subsequence

What do we need to know to decide on element i?

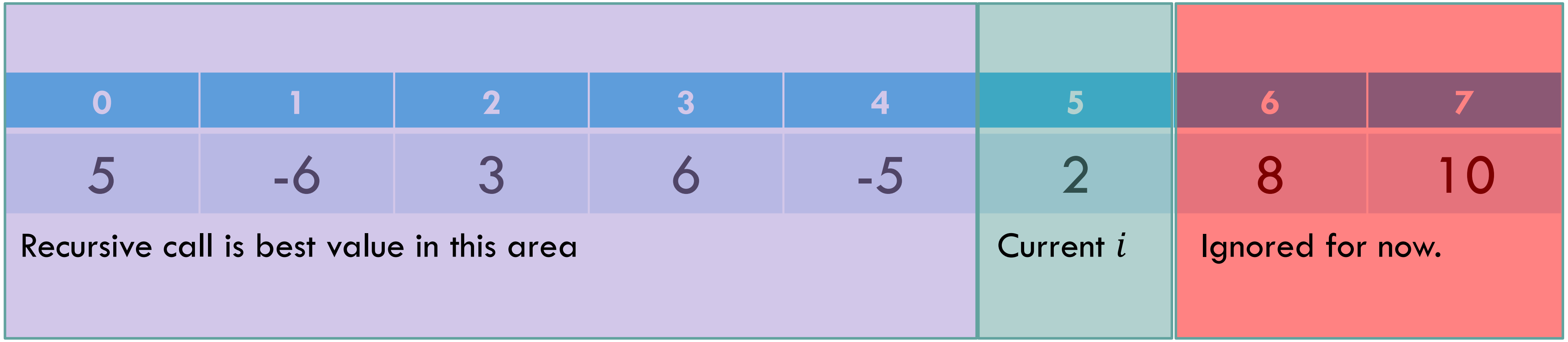
Is it allowed?

Will the sequence still be increasing if it's included?

Still thinking right to left --

Two indices: index we're looking at, and index of upper bound on elements (i.e. the value we need to decide if we're still increasing).

Recurrence



0	1	2	3	4	5	6	7
5	-6	3	6	-5	2	8	10

Recursive call is best value in this area

Current i

Ignored for now.

Need recursive answer to the left

Currently processing i

Recursive calls to the left are needed to know optimum from $1 \ldots i$

Will move i to the right in our iterative algorithm

Longest Increasing Subsequence

$LIS(i, j)$ is “Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$”

Need a recurrence

$$LIS(i, j) = \begin{cases} ? & \text{if } i < 0 \\ ? & \text{if } i = 0 \\ ? & \text{if } A[i] > A[j] \\ ? & \text{otherwise} \end{cases}$$

Longest Increasing Subsequence

$LIS(i, j)$ is “Number of elements of the maximum increasing subsequence from $0, \dots, i$ where every element of the sequence is at most $A[j]$”

Need a recurrence

$$LIS(i, j) = \begin{cases} ? & \text{if } i < 0 \\ ? & \text{if } i = 0 \\ ? & \text{if } A[i] > A[j] \\ ? & \text{otherwise} \end{cases}$$

If $A[i] > A[j]$ element i cannot be included in an increasing subsequence where every element is at most $A[j]$. So taking the largest among the first $i - 1$ suffices.

If $A[i] \leq A[j]$, then if we include i, we may include elements to the left only if they are less than $A[i]$ (since $A[i]$ will now be the last, and therefore largest, of elements $1 \dots i$. If we don’t include i we want the maximum increasing subsequence among $1 \dots i - 1$.

Longest Increasing Subsequence

$LIS(i, j)$ is "Number of elements of the maximum increasing subsequence from $0, \dots, i$ where every element of the sequence is at most $A[j]$"

Need a recurrence

$$LIS(i, j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1, j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1, i), LIS(i-1, j)\} & \text{otherwise} \end{cases}$$

If $A[i] > A[j]$ element i cannot be included in an increasing subsequence where every element is at most $A[j]$. So taking the largest among the first $i - 1$ suffices.

If $A[i] \leq A[j]$, then if we include i, we may include elements to the left only if they are less than $A[i]$ (since $A[i]$ will now be the last, and therefore largest, of elements $0 \dots i$. If we don't include i we want the maximum increasing subsequence among $0 \dots i - 1$.

Longest Increasing Subsequence

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

Memoization structure? $n \times n$ array.

Filling order?

LIS

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j

i

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5								
1, −6								
2, 3								
3, 6			🟫		🟫			
4, −5			🟪					
5, 2								
6, 8								
7, 10								

LIS

$LIS(1,0)$ $A[1] < A[0]$ not allowed:
Take $LIS(0,0)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \le A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j

i

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1							
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$LIS(1,1)$ $A[1] \leq A[1]$ can add, $1 + LIS(0,1)$ or $LIS(0,1)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j

i

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1						
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$LIS(1{,}2)$ $A[1] \leq A[2]$ allowed to add:
$1 + LIS(0{,}1)$ or $LIS(0{,}2)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j

i

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1	1					
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$LIS(2{,}0)$ $A[2] \leq A[0]$ allowed to add
$1 + LIS(1{,}2)$ or $LIS(1{,}0)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j

i

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1	1	1	1	1	1	1
2, 3	2							
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j

i

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1	1	1	1	1	1	1
2, 3	2	1	2	2	1	1	2	2
3, 6	2	1	2	3	1	1	3	3
4, −5	2	1	2	3	2	2	3	3
5, 2	3	1	3	3	2	3	3	3
6, 8	3	1	3	3	2	3	4	4
7, 10	3	1	3	3	2	3	4	5

pseudocode

```
//real code snippet that actually generated the table on the last slide
for(int j=0; j < n; j++){
     vals[0][j] = (A[0] <= A[j]) ? 1 : 0;
}
for(int i = 1; i < 8; i++){
     for(int j = 0; j < n; j++){
           if(A[i] > A[j])
                 vals[i][j] = vals[i-1][j];
           else{
                 vals[i][j] = Math.max(1+vals[i-1][i], vals[i-1][j]);
           }
     }
}
```

Longest Increasing Subsequence

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

Memoization structure? $n \times n$ array.

Filling order?

Outer loop: increasing i

Inner loop: increasing j

LIS

One more thing….what's the final answer?

We want the longest increasing sequence in the whole array.

$LIS(i, j)$ is "Number of elements of the maximum increasing subsequence from $0, \dots, i$ where every element of the sequence is at most $A[j]$"

What do we want?

LIS

One more thing….what’s the final answer?

We want the longest increasing sequence in the whole array.

$LIS(i, j)$ is “Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$”

$\max_j LIS(n, j)$. Intuitively, j represents “the last element” in the array. Anything could be the last one! Take the maximum.