Even More Dynamic Programming

Closing the Loop on Feedback

Got a summary of feedback from Ken – thank you for taking that time:

Thanks for talking to Ken live or filling out the form

What's working:

Ed Q&A

Office Hours (when they're not overwhelmed)

Lecture Activities (a little less for those asynchronous)

Closing the Loop

Changes:

Homeworks were too long

-HW1 was WAY too long

-HW2 was still too long (or at least problem 3 needed a little more scaffolding). Particularly since instead of being 2 weeks, it was only 1.5 (or 1 if you used late days on HW1).

Going forward: fewer problems

Weeks that an ethics project is due, one fewer problem on main hw.

Closing the Loop

Hard to know required level of detail on proofs

-having sample solutions may help.

-On some problems listing length of our solutions,

-On some problems, scaffolding a possible proof

-Also, style guide to record general advice.

-we know it's hard (it's hard even at the end of a 10-week course on it); we're going to try to be lenient.

On HW2, input/output
More examples of input/output. (hopefully starting on HW4)

One more request/advice

If you aren't already…start early on the homework.

I know. Everyone says this.

Algorithms questions benefit tremendously from spaced out thinking (the back of your brain/your sleeping brain does magic)

You'll need less active thinking time if you spread out over multiple days.

You'll also spread out office hour demand ☺

Closing the Loop

Gap between high level of lecture and actual code

-Trying to add pseudocode to more lecture examples

-Any of the optional textbooks (including the free online one – algorithms.wtf) will have pseudocode.

-More practice on homework with that.

Edit Distance

Given two strings x and y, we'd like to tell how close they are.

Applications?

Spelling suggestions

DNA comparison

Edit Distance

More formally:

The edit distance between two strings is:

The minimum number of **deletions, insertions,** and **substitutions** to transform string x into string y.

Deletion: removing one character

Insertion: inserting one character (at any point in the string)

Substitution: replacing one character with one other.

Example

What's the distance between `babyyodas` and `tastysoda`?

B	A	B		Y	Y	O	D	A	S
sub		sub	ins		sub				del
T	A	S	T	Y	S	O	D	A	

Distance: 5, one point for each colored box

Quick Checks – can you explain these?
If x has length n and y has length m, the edit distance is at most $\max(x, y)$

The distance from x to y is the same as from y to x (i.e. transforming x to y and y to x are the same)

Finding a recurrence

What information would let us simplify the problem?

What would let us "take one step" toward the solution?

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

$OPT(i, j)$ is the edit distance of the strings $x_1 x_2 \cdots x_i$ and $y_1 y_2 \cdots y_j$. (we're indexing strings from 1, it'll make things a little prettier).

The recurrence

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

Write a recurrence.

What do we need to keep track of? Where we are in each string!

Match right to left – be sure to keep track of characters remaining in each string!

The recurrence

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

What does delete look like? $OPT(i-1, j)$ (delete character from x match the rest)

Insert $OPT(i, j-1)$ Substitution: $OPT(i-1, j-1)$

Matching charcters? Also $OPT(i-1, j-1)$ but only if $x_i = y_j$

The recurrence

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

Delete

Insert

Sub and matching

$$OPT(i,j) = \begin{cases} \min\{\, 1 + OPT(i-1,j),\ 1 + OPT(i,j-1),\ \mathbb{I}[x_i \neq y_j] + OPT(i-1,j-1)\} \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

"Indicator" – math for "cast bool to int"

Dynamic Programming Process

1. Define the object you're looking for

   Minimum Edit Distance between x and y

2. Write a recurrence to say how to find it

3. Design a memoization structure

4. Write an iterative algorithm

Memoization

$$OPT(i,j) = \begin{cases} \min\{\, 1 + OPT(i-1,j), 1 + OPT(i,j-1),\ \mathbb{I}[x_i \neq y_j] + OPT(i-1,j-1)\} \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

2D array n by m

`OPT[i][j]` is $OPT(i,j)$

Edit Distance

$\mathbf{OPT}(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0										
T 1										
A 2										
S 3										
T 4										
Y 5										
S 6										
O 7										
D 8										
A 9										

Edit Distance

$OPT(i,j)$		0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0		0	1	2	3	4	5	6	7	8	9
T	1	1									
A	2	2									
S	3	3									
T	4	4									
Y	5	5									
S	6	6									
O	7	7									
D	8	8									
A	9	9									

Edit Distance

$OPT(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1								
A 2	2		$1+1\ (left, delete)$							
S 3	3		$1+1(up, insert)$							
T 4	4		$0+1(diag, sub)$							
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

$OPT(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1							
S 3	3									
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

$2 + 1\ (left, delete)$

$2 + 1(up, insert)$

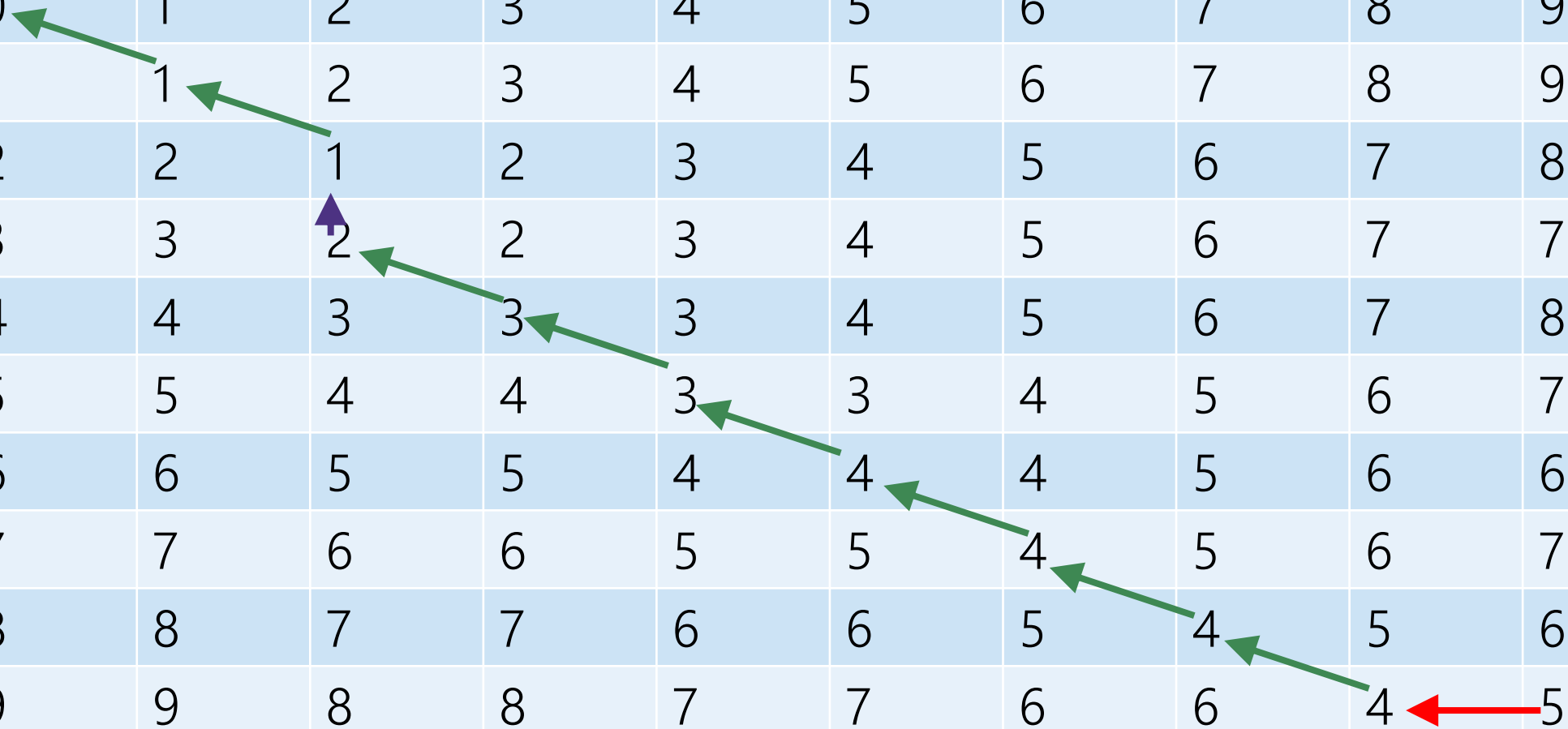
$1 + 0(diag, sub)$

Edit Distance

$OPT(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5

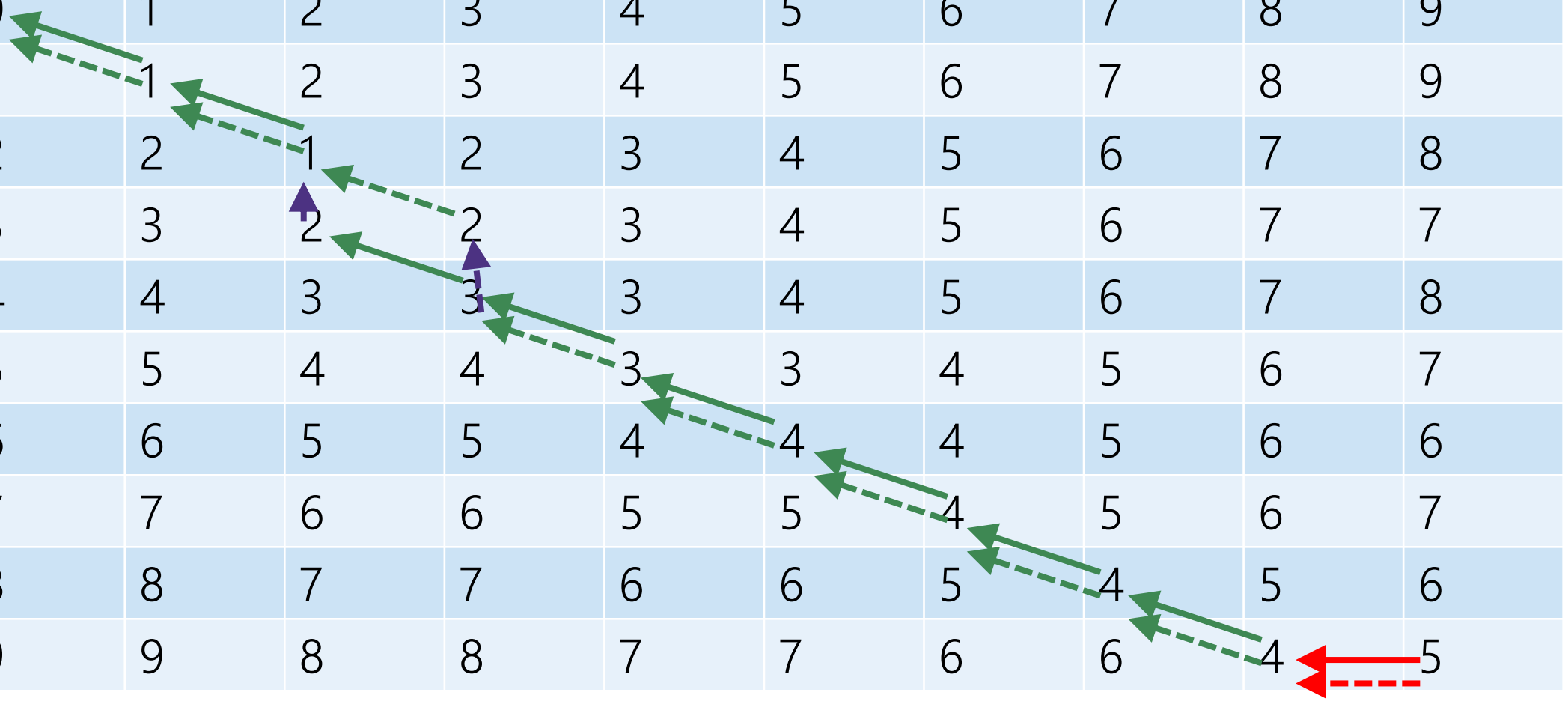
Edit Distance

$OPT(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5



Edit Distance

$OPT(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5



Dynamic Programming Process

1. Define the object you're looking for

   Minimum Edit Distance between x and y

2. Write a recurrence to say how to find it

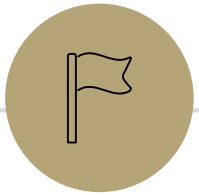
   ✅

3. Design a memoization structure

   $m \times n$ Array

4. Write an iterative algorithm

   Outer loop: increasing rows (starting from 1)
   Inner loop: increasing column (starting from 1)



More Problems

Maximum 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 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.

Is it enough to know OPT(i)?

Trying to Recurse

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 $i-1$**

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

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Need two recursive values:

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

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							

$INCLUDE(i)$

0	1	2	3	4	5	6	7
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

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	5	7	7	7	7	10

$INCLUDE(i)$

0	1	2	3	4	5	6	7
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]
```

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

4 is optimal for the array above

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

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 min to its right (i.e. the value we need to decide if we're still increasing).

Longest Increasing Subsequence

$LIS(i,j)$ is "Number of elements of the maximum increasing subsequence from $1,\dots,i$ where every element of the sequence is at most $A[j]$"

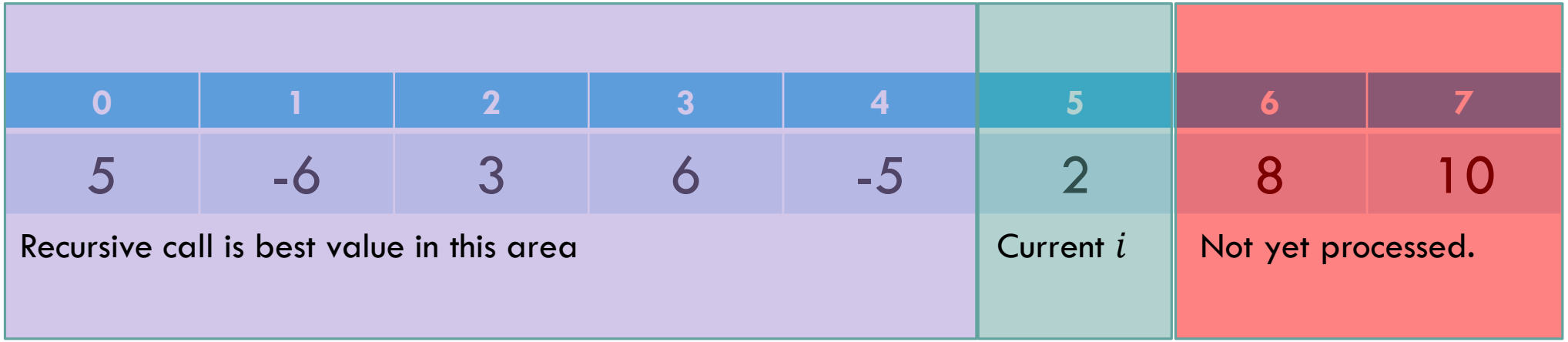
Need a recurrence

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

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 $A[i] \le A[j]$, 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 $1 \dots i-1$.

Recurrence



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

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

Memoization structure? $n \times n$ array.

Filling order? Multiple possible

Outer loop: j from 0 to $\mathrm{n} - 1$ OR from min $A[j]$ to max $A[j]$

Inner loop: i from 0 to $n - 1$

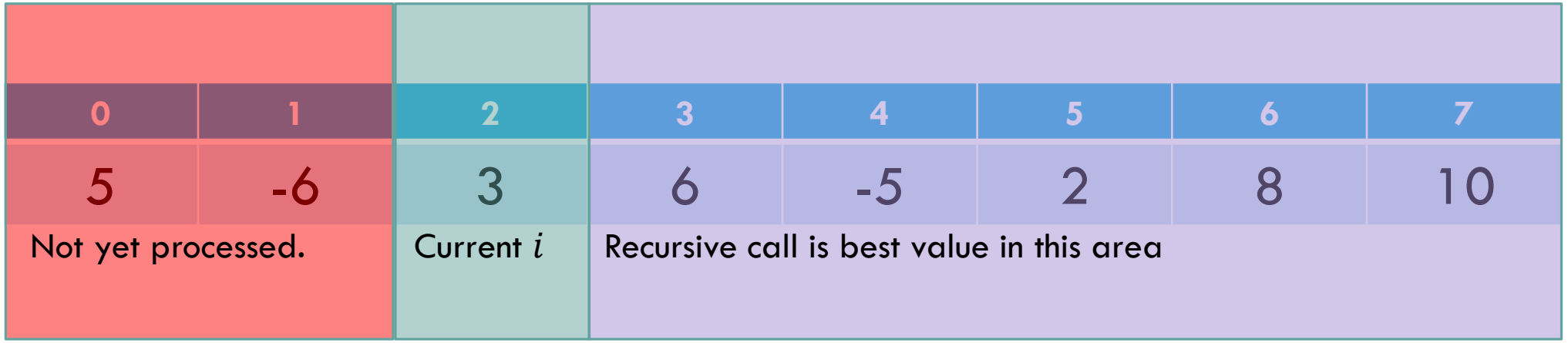
Longest Increasing Subsequence

Think left-to-right instead of right-to-left

$LISAlt(i, j)$ is "Number of elements of the maximum increasing subsequence from $i, \ldots, n$ where smallest element of the sequence is $A[j]$"

$$LISAlt(i,j) = \begin{cases} 0 & \text{if } i > n \text{ or } j > n \\ LIS(i+1, j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i+1, i), LIS(i+1, j)\} & \text{o/w} \end{cases}$$

Recurrence



Need recursive answer to the right

Currently processing i

Recursive calls to the right are needed to know optimum from $i \ldots n$

Will move i to the left in our iterative algorithm

Longest Increasing Subsequence

$LISAlt(i, j)$ is "Number of elements of the maximum increasing subsequence from $i, \ldots, n$ where smallest element of the sequence is $A[j]$"

$$LISAlt(i,j) = \begin{cases} 0 & \text{if } i > n \text{ or } j > n \\ LIS(i+1, j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i+1, i), LIS(i+1, j)\} & \text{o/w} \end{cases}$$

Memoization structure? $n \times n$ array.

Filling order? Multiple possible

Outer loop: j from $n-1$ to 0

Inner loop: i from 0 to $n-1$

Summing Up

The two recurrences have the same idea (add/don't add, record the end of the array closest to your next decision)

But thinking left-to-right vs. right-to-left

Both end up with an $n \times n$ memoization structure

And $O(n^2)$ running time.

But Wait! There's more

Another recurrence at the end of these slides for more practice.

Instead of thinking "do I include this element or not?" for each element,

Ask "what's the next element" or equivalently "what's the longest subsequence starting from me"

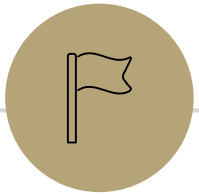
Get a different recurrence, but not a better running time.

Takeaways

When designing a dynamic program, we sometimes need to introduce a second variable, that doesn't appear in the program

Or a second recurrence that mixes with the first if other decisions affect what's optimal (beyond which problem you look at)

There might be more than one program available.



Extra Practice

Subset Sum

Given an array $A[]$ of positive integers, and a number t find whether there is a subset of $A[]$ that sums to exactly t.

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

If $t = 30$, answer is "yes" (for example, $5 + 5 + 2 + 8 + 10$)

If $t = 100$, answer is "no" (not allowed to repeat elements beyond the number of copies in the array, e.g. can't say "10 copies of 10")

Subset Sum

Write an English description of what you want to calculate

Write a recurrence

Give a sentence or two (in English) of why your recurrence should work.

Subset Sum

Write an English description of what you want to calculate

Let $SUBSUM(i, t)$ be true if and only if a subset of $A[0], \ldots, A[i]$ can sum to t.

Write a recurrence

$$SUBSUM(i,t) = \begin{cases} True & \text{if } t = 0 \\ False & \text{if } i < 0 \text{ and } t \neq 0 \\ SUBSUM(i-1,t) || SUBSUM(i-1, t - A[i]) & o/w \end{cases}$$

Give a sentence or two (in English) of why your recurrence should work.

Element i is either included or it isn't – if i appears in a valid subset, then we need to have the remaining elements sum to $t - A[i]$. If i doesn't appear then the remaining elements will get to t. We "or" together because either could be a valid path to

Subset Sum

What memorization structure will you use?

A 2D Boolean array SUBSUM(i, j). Array will be $n \times T$

Write the pseudocode to fill up the structure iteratively.

```
SubSum(int[] A, int T)
Bool[][] SubSum = new Bool[n][T+1]
for(int j=0;j<T+1;j++){ SubSum[0][j]=False;}
SubSum[0][A[0]]=True;
for(int i=1; i<n;i++){
  for(int j=0; j<T+1; j++){
     if(SubSum[i-1][j]){
       SubSum[i][j]=True;
       SubSum[i][j+A[i]]=True;//need to catch Array index errors. Don't do
                              //this in real code.
     }
  }
}
return SubSum[n][T-1];
```

Longest Increasing Subsequence, Round 3

Let's ask "what's the best choice for the next element" (instead of just "is this the next element"

What's the best choice?

It has to be greater than our current element, after that it's the one that can lead to the longest subsequence.

So, (since we're starting with our current element), the question is

"what's the longest increasing subsequence, starting at index i"

Longest Increasing Subsequence, Round 3

Let $LISStart(i)$ be the length of the longest increasing subsequence among indices $i \ldots n$, that starts at index i.

Call an index "valid" if $A[j] > A[i]$ (it's "valid" to add j to a sequence starting at i

$$LISStart(i) = \max\{1, \max_{j:j \text{ is valid and } j>i} \{LISStart(i)\} \text{ if } i \leq n\} \}$$

i.e. have a single entry (yourself) or prepend yourself to the longest subsequence starting after you (that you can prepend yourself to)

Longest Increasing Subsequence, Round 3

Memoization? 1D array of size n

Iteration? Outer-loop: i decreasing

Inner-loop: calculate $LISStart(i)$ by iterating over previous calculations.

Checking n values for each new calculation, not $O(1)$

Still $O(n^2)$ time.

Be careful!

Final answer is **not** $LISStart(i)$.

It's the maximum entry among $LISStart()$ array