DP4
More Arrays and Strings

CSE 421 26Sp
Lecture 14

Announcements

Don't forget to fill out the exam conflict/handedness form for midterm 1.

Was due at noon, but we've left it open until lecture ends.

Update to the topic coverage for the midterm:
Divide and conquer can show up **only** in multiple choice/short answer questions (no big "design an algorithm" questions).

Previously, big questions for D&C were fair game

We might have them on midterm 2 instead.

Longest Increasing Subsequence

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

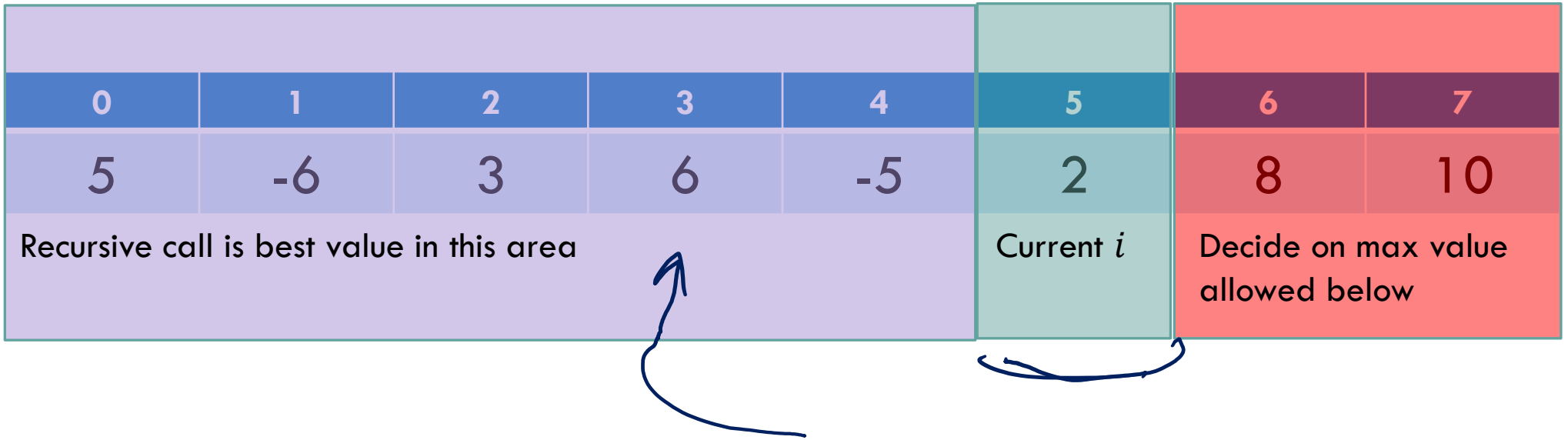
Longest set of (not necessarily consecutive) elements that are increasing

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.

LIS, visualized



Need recursive answer to the left

Currently processing i

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

LIS Recurrence

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

"Indicator" kind of like 312: cast bool to int.

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 $0 \ldots i$. If we don't include i we want the maximum increasing subsequence among $0 \ldots i - 1$.

Finishing LIS

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

Memoization structure? $n \times n$ array.

Filling order?

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]);
          }
     }
}
```

LIS (1)

0	1	2	3
5	−6	3	6

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

j

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

i

The recursive call that made us included $A[6] = 8$ in the sequence.
We now need to decide on $A[3] = 6$.
$A[3] = 6 < 8 = A[6]$. We are allowed to include $A[3]$.
Try that: then need LIS(2,3) [LIS among A[0,…,2] where all less than A[3]
And try LIS(2,6) [don't include] LIS among A[0,..,2] all less than A[6]

LIS (2)

0	1	2	3
5	−6	3	6

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

j →

	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

i ↓

LIS (3)

0	1	2	3
5	−6	3	6

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

j

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

i

The recursive call that made us included $A[5]$ is the sequence.
We now need to decide on $A[3] = 6$.
$A[3] = 6 > 2 = A[5]$. We **cannot** include $A[3]$.
Need LIS among A[0,…,2] with

LIS (4)

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

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

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

LIS (5)

$LIS(1,1)$ $A[1] \leq A[1]$ 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 →

	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								

i ↓

LIS (6)

$LIS(1,2)$ $A[1] \leq A[2]$ 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 (7)

$LIS(2,0)$ $A[2] \leq A[0]$ 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 (8)

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

j →

	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

i ↓

Finishing LIS (iterative order)

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

Memoization structure? $n \times n$ array.

Filling order?

- Outer loop: increasing i
- Inner loop: increasing j

Getting the Final Answer

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 $A[j]$"

What do we want?

Final Answer (version 1)

The principled approach:

What does j mean? It's an already added element to our right.

There's nothing already on our right at the start. In recursive code, we'd probably call $LIS(n, \texttt{null})$. So do that…just tweak the recurrence

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

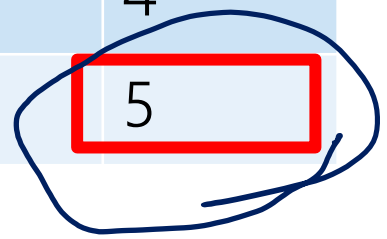
LIS, final answer

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

j →

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

i ↓



Final Answer (option 2)

The clever hack:

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

So, if j is the last element of the true sequence, that's what we want!

Just return $\max_j \text{LIS}(n, j)$

LIS

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

j

	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

i

Final Answer (option 2, explained)

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.

Takeaways

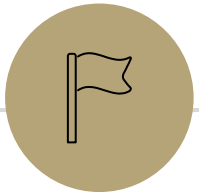
Sometimes you need to add an extra recurrence

Or add an extra parameter.

You need to write a recursive function!

That's all DP is.

But the recursion itself can be tricky.



More Practice: Edit Distance

Edit Distance

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`?

x	B	A	B		Y	Y	O	D	A	S
op	Sub		sub	ins		sub				del
y	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 minimum number of insertions, deletions, and substitutions to transform $x_1 x_2 \cdots x_i$ into $y_1 y_2 \cdots y_j$. (we're indexing strings from 1, it'll make things a little prettier).

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

The recurrence (starting)

"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.

$$OPT(i,j) = \begin{cases} \min\{\ 1 + OPT(i-1,j), 1 + OPT(i,j-1),\ 1 + OPT(i-1,j-1), \} \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

Delete

Insert

Substitution

TODO: Just Match

The recurrence (v1, we'll improve soon)

"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.

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

Delete | Insert | Substitution | Just Match

Idea: only allow "just match" when you can just match.
Otherwise make it ∞ (will never be the min).
In code: if/else branch, probably. This is a math notation trick.

"Indicator" like from 312

The recurrence (finalized)

"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

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

When we could match, we will never substitute; matching will always give us a better score! Still have to check delete, insert (those could be better).

Computing (1)

$\mathbf{OPT}(i,j)$	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										

Computing (2)

$\mathbf{OPT}(i,j)$		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									

Computing (3)

x	B	A	B	Y
y	T	A	S	

Current subproblem: edit dist between BABY and TAS

$\mathrm{OPT}(i,j)$	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									

Computing (4)

x	B	A	B	Y
y	T	A	S	

Delete: get rid of Y (cost 1)
BAB
TAS

$OPT(i,j)$	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									

Computing (5)

x	B	A	B	Y
y	T	A	S	

Insert: insert S (cost 1)
BABY
TA

$OPT(i,j)$	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									

Computing (6)

x	B	A	B	Y
y	T	A	S	

Sub: Transform Y to S (cost 1)
BAB
TA

$\mathbf{OPT}(i,j)$	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									

Computing (7)

x	B	A	B	Y
y	T	A	S	

Gold entry will be min of:

$1 +$ delete

$1 +$ insert

$1 +$ sub

$OPT(i,j)$	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									

Computing (8)

$1 + 2$
$1 + 3$
$1 + 2$
Min: 3

Gold entry will be min of:
$1 +$ delete
$1 +$ insert
$1 +$ sub

$OPT(i,j$	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					
S 6										
O 7										
D 8										
A 9										

Computing (9)

Fill in the next two entries. Be careful with the sub/match distinction!

$\mathbf{OPT}(i,j)$		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					
S	6										
O	7										
D	8										
A	9										

Computing (10)

Fill in the next two entries. Be careful with the sub/match distinction!

$OPT(i,j)$	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**			
S 6										
O 7										
D 8										
A 9										

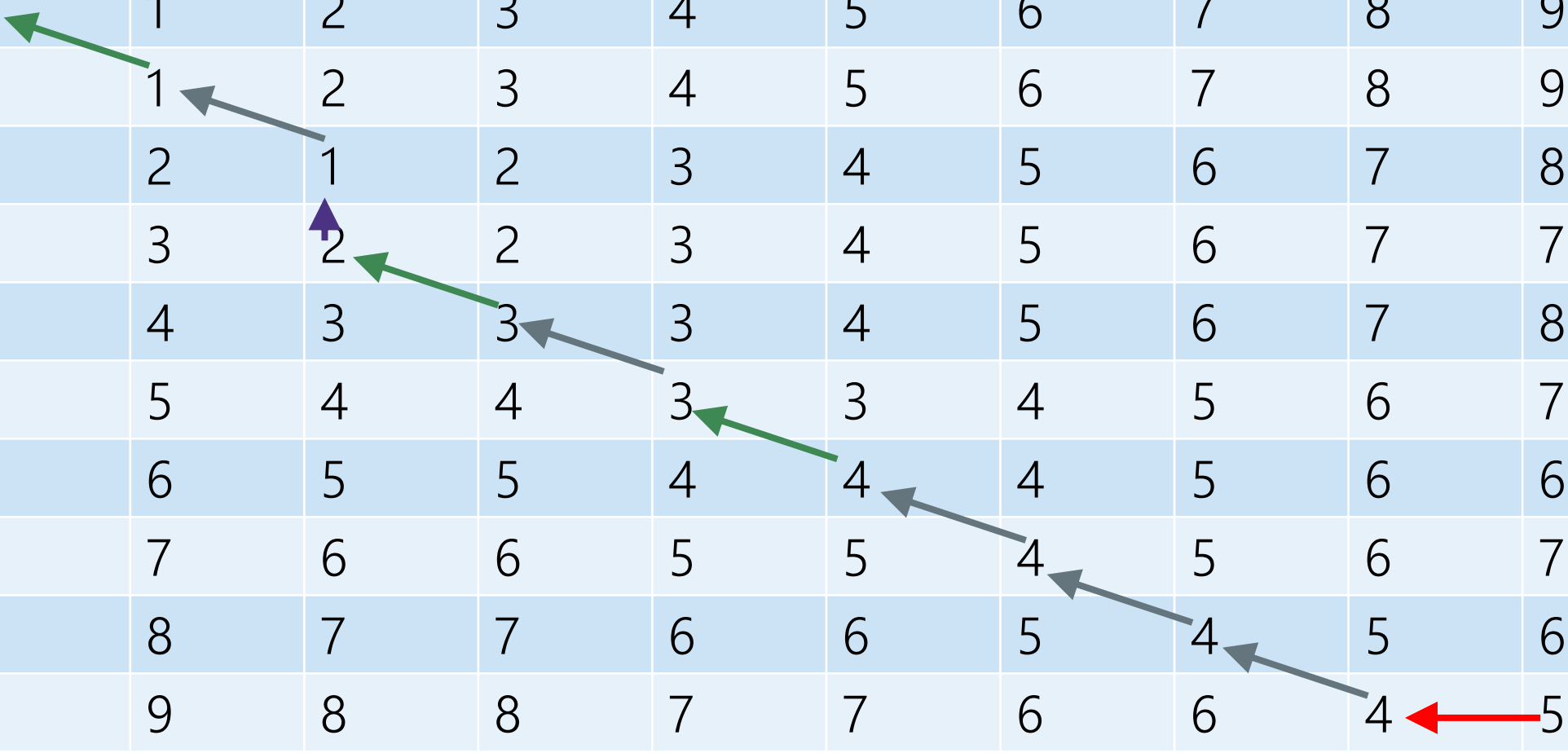
Y's match, so sub is free!

Computing (11)

$\mathbf{OPT}(i,j)$	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

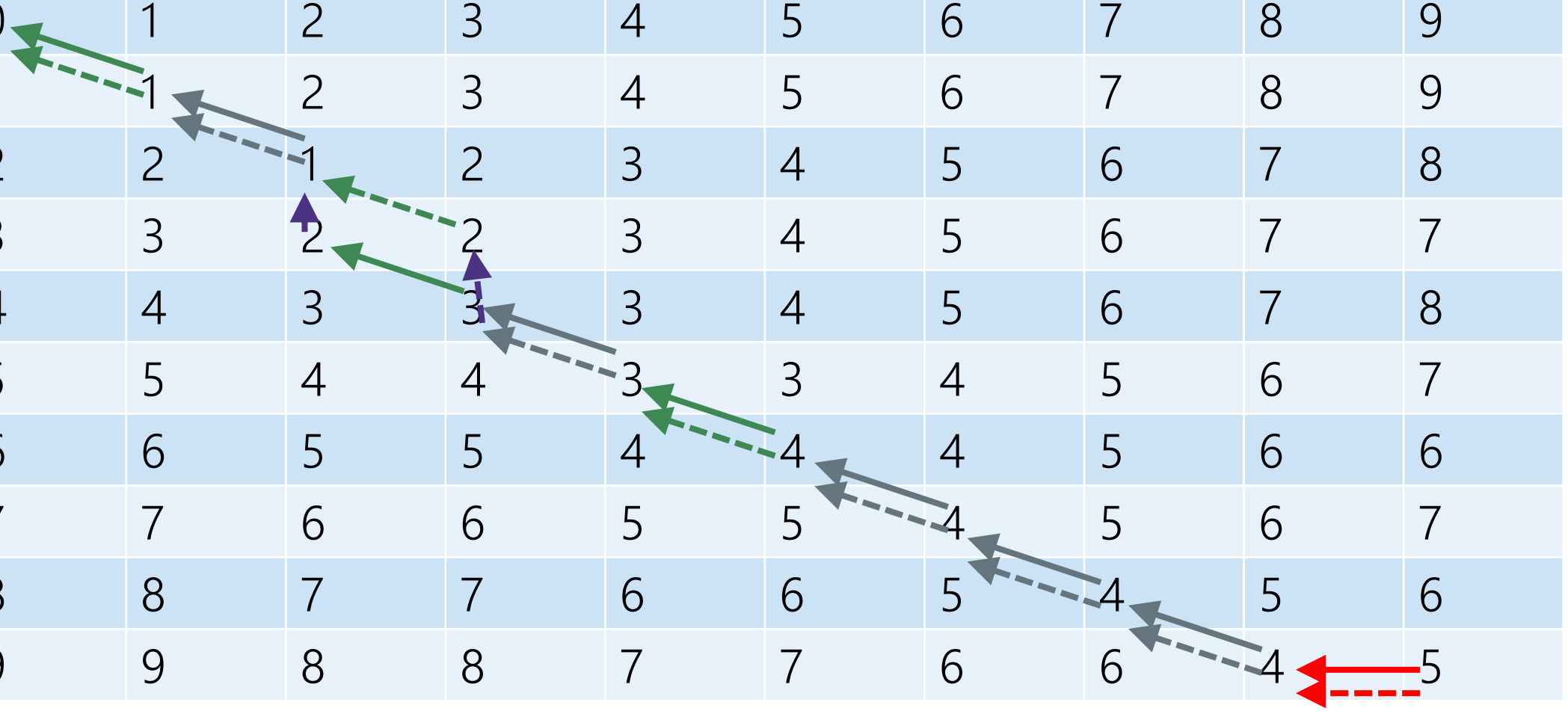
What operations?

$OPT(i,j)$	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



What operations? Many options

$OPT(i,j)$	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

   $OPT(i, j)$ is the minimum number of insertions, deletions, and substitutions to transform $x_1 x_2 \cdots x_i$ into $y_1 y_2 \cdots y_j$.

2. Write a recurrence to say how to find it

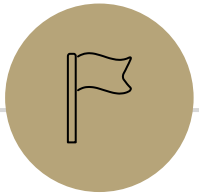
   ✅

3. Design a memoization structure

   $m \times n$ Array

4. Write an iterative algorithm

   Outer loop: increasing j (starting from 1)
   Inner loop: increasing i (starting from 1)



DP Thought Process

Goal of DP

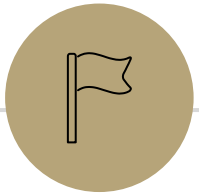
Just try all the (reasonable) possibilities.

Don't worry about greedily choosing the best, use recursion to "look ahead" for all the best options, and pick the best one.

There is a "greedy-ish" alteration to the Edit Distance recurrence…

It turns out, if the two characters match, that will always be at least as good as the insert/delete options.

But it's fine to **not** notice! And if you thought it was safe but wasn't, well….



DP running times

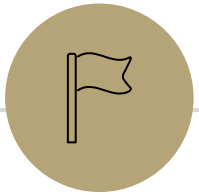
Analyzing Running Times

We haven't analyzed the running times of the DP algorithms we've done so far very closely.

Usually much easier with the iterative version than recursive.

Once you make the switch, the code is usually "just" a few levels of embedded for-loops, each doing constant work.

But be careful---sometimes you take a max of n things, which takes $\Theta(n)$ time.



Proofs of DP algorithms

DP Proofs (1)

We generally **won't** ask you for proofs of correctness on dynamic programming problems. (maybe one after the midterm)

Why?

The proofs are always inductive proofs where you say

"my recurrence checks all the possibilities" or, equivalently

"The maximum thing has to be made up of the best thing for all these other subproblems."

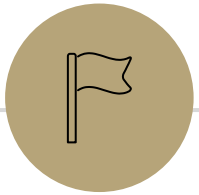
The proof itself is a lot of boilerplate and hard to write clearly (you have to differentiate between your recurrence and what your recurrence intends to calculate, which can be tricky).

DP Proofs (2)

There's one example proof sometime in the next few week so you know what you're (not) missing.

Instead, we're going to ask for your intuition on what your recurrence is doing (what do all the cases correspond to/why are they exhaustive)?

The proof is just a lot of formalism on that key idea. So we're going to have you focus on the idea, not the formalism.



A Sample Proof: LIS

What would a proof look like? (1)

$P(i)$: for all j, $LIS(i, j)$ calculates the length of the longest increasing subsequence among elements $0, \ldots, i$, where every element is at most $A[j]$.

We argue by induction on i.

Base Case: Let $i = 0$

$LIS(0, j)$ (by the recurrence) is $\mathbb{I}[A[0] \leq A[j]]$. Among elements $0, \ldots, 0$, the only sequences are the empty sequence and the single element $A[0]$. If $A[0] \leq A[j]$, then the one element is allowed, and we return 1.

If $A[0] > A[j]$, then no elements are less than $A[j]$ and the empty sequence is the only legal one. 0 is correctly returned as $A[0] > A[j]$.

What would a proof look like? (2)

IH: Suppose $LIS(i, j)$ calculates the length of the longest increasing subsequence among elements $0, \ldots, i$ where every element is at most $A[j]$, for all j and for $i = 0,1, \ldots, k$ (k arbitrary, $k \geq 0$)

IS: Consider $LIS(k + 1, j)$

Case 1: $A[k + 1] \leq A[j]$
The longest increasing subsequence among elements $0, \ldots, k + 1$ with all elements at most $A[j]$ either includes or excludes $A[k + 1]$.

Case 1A:
If $A[k + 1]$ is included, then the rest of the sequence is an increasing subsequence among $0, \ldots, k$ with each element less than $A[k + 1]$. By inductive hypothesis that is equivalent to $LIS(k, k + 1)$.
The value $1 + LIS(k, k + 1) = LIS(k + 1, j)$ is therefore correct.

What would a proof look like? (3)

IH: Suppose $LIS(i, j)$ calculates the length of the longest increasing subsequence among elements $0, \ldots, i$ where every element is at most $A[j]$, for all j and for $i = 0,1, \ldots, k$ (k arbitrary, $k \geq 0$)

IS: Consider $LIS(k + 1, j)$

Case 1B: $A[k + 1]$ is excluded
If $A[k + 1]$ is excluded, then the rest of the sequence is an increasing subsequence among $0, \ldots, k$ with each element still less than $A[j]$. By inductive hypothesis that is equivalent to $LIS(k, j)$.

In both cases, the recursive calls represent lengths of valid sequences, so taking the max gives us the longest increasing subsequence.

What would a proof look like? (4)

IH: Suppose $LIS(i, j)$ calculates the length of the longest increasing subsequence among elements $0, \dots, i$ where every element is at most $A[j]$, for all j and for $i = 0,1, \dots, k$ (k arbitrary, $k \geq 0$)

IS: Consider $LIS(k + 1, j)$

Case 2: $A[k + 1] > A[j]$

Matches case 1B ($A[k + 1]$ is too large to be allowed in the sequence so must be excluded).

Key Insights

There's really two key claims in the proof:

1. We're checking all the reasonable options ($A[k+1]$ is either included or excluded)

2. The LIS really can be built from the smaller LISs.
i.e. you can't get a longer sequence other than those found recursively.

That wouldn't be true if you didn't have the j parameter (or do another clever thing---see the section solutions). The LIS among $0, \ldots, k$ might not be made from LIS among $0, \ldots, k-1$

Consider [1,2,100,3,4,5] LIS(2) is 1,2,100; LIS(3) is 1,2,100 {skip 3}; LIS(4) is 1,2,3,4.

The boilerplate:insight ratio is very high. We just ask you to give us the insight on HW5.