Algorithms

More Dynamic Programming

Recap

- HW2 - done today
- HW3 - written, in groups due a week from Monday

Looking forward!

Oral grading likely on Oct 8

Midterm likely on 11th or 14th

Steps:

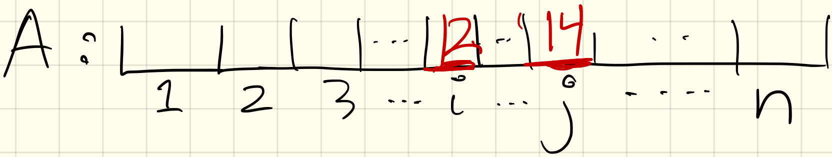
1. Formulate the recursion
2. Build solution from base case up.
   - identify subproblems
   - identify dependencies:
     ie: F(6) depends on F(5) + F(4)
   - choose data structure
     ie: often array, 1d or 2d, or even a few variables
   - choose evaluation order
   - write pseudo code, then analyze time/space

Let's look at an old friend or two...

Back to LIS:

Some notation: LISHelper in book

Let $LIS(i,j)$:= length of longest subsequence of $A[j..n]$ with elements $> A[i]$



Then:

$$LIS(i,j) = \begin{cases} 0 & \text{if } j > n \\ LIS(i, j+1) \leftarrow \text{Skip } j & \text{if } A[i] \geq A[j] \\ \max\{LIS(i, j+1),\ 1 + LIS(j, j+1)\} & \text{otherwise} \end{cases}$$

helper

↳ could add $A[j]$ so try with & without including

$$LIS(A[1,..,n]) = \left[\begin{array}{l} \text{add } -\infty \text{ as } A[0], \\ \text{Call LISHelper}(0,1) \end{array}\right] - 1$$

What are my dependencies?

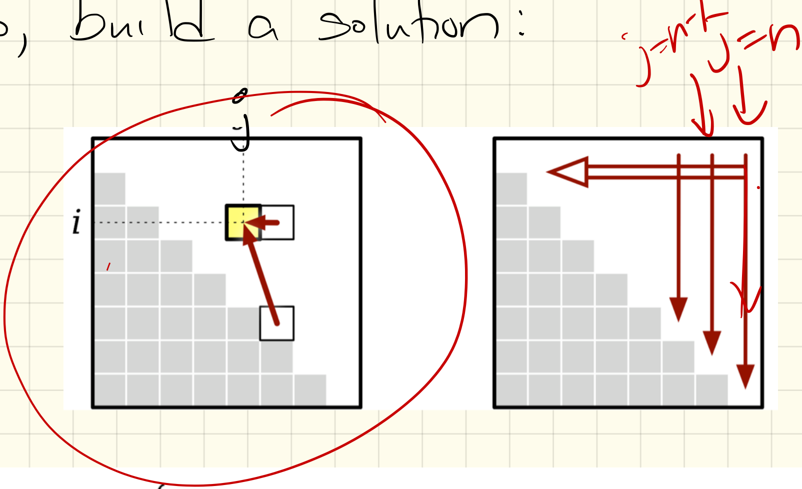
Note:

LISHelper(i, j)
depends on:

- LISHelper(i, j+1)
- LISHelper(j, j+1)

Order: better start at (n, n)

So, build a solution:



$$LIS(i,j) = \begin{cases} 0 & \text{if } j > n \\ LIS(i, j+1) & \text{if } A[i] \geq A[j] \\ \max\{LIS(i, j+1),\ 1 + LIS(j, j+1)\} & \text{otherwise} \end{cases}$$

Observe:

for $j \leftarrow n$ down to 1

fill in column j

for $i \leftarrow 1$ to $j-1$

// entry (i,j) needs $(i,j+1)$ & $(j,j+1)$

Algorithm:

```
LIS(A[1..n]):
    A[0] ← −∞                            ⟨⟨Add a sentinel⟩⟩
    for i ← 0 to n                       ⟨⟨Base cases⟩⟩
        LIS[i, n+1] ← 0

    for j ← n downto 1
        for i ← 0 to j − 1
            if A[i] ≥ A[j]
                LIS[i, j] ← LIS[i, j+1]      O(1)
   O(1)     else
                LIS[i, j] ← max{LIS[i, j+1], 1 + LIS[j, j+1]}
    return LIS[0, 1]
```

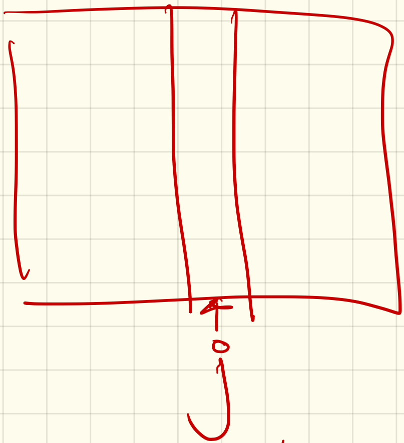
Time & Space:

Runtime: $\sum_{j=1}^{n}\left[\sum_{i=1}^{j} O(1)\right]$

$= \sum_{j=1}^{n} j = O(n^2)$

Space: $n \times n$ array $= O(n^2)$

Next: consider space again



To fill in column j, need column j+1

So store 2 arrays, prev + current

↳ $O(n)$ space

(Need $O(n^2)$ space to store sequences.

Length needs $O(n)$)

Edit Distance

The minimum number of deletions, insertions, or substitutions of letters to transform between two strings.

Ex: F O O D

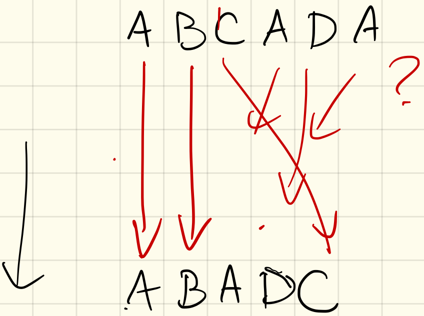
+1 +1 +1 insert +1

M O N E Y

Uses? - Spell checker
- bioinformatics

Don't be greedy!

The temptation is to do this as you go:



edit distance?

Idea: try matching, or not

try both, pay costs that depend on letters

How to solve:

Aligning/matching will help:

A: A L G O R   I   T H M

+0   +0   delete

B: A L   T R U I S T I C

+1   +1   +1   +1   +1   +1

distance?

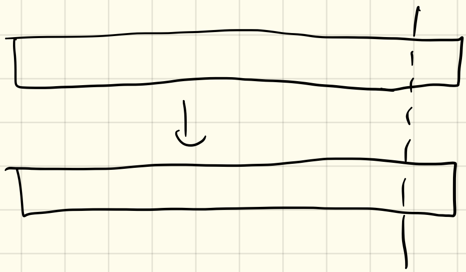
6

Recursive formulation:

If I align like this, can observe:

If you delete last (aligned) column, the rest will still be optimal for shorter substrings edit distance.

Why?



Turning this into a matrix:

Let $EDIT(A[1..m], B[1..n])$ be edit distance b/t A & B.

When we choose how to align, 3 possibilities:

- insertion:

- deletion:

- substitution:

Turn this into recursion:

$$\boxed{Edit(A[1\,..\,m], B[1\,..\,n]) = \min \left\{ \begin{array}{l} Edit(A[1\,..\,m-1], B[1\,..\,n]) + 1 \\ Edit(A[1\,..\,m], B[1\,..\,n-1]) + 1 \\ Edit(A[1\,..\,m-1], B[1\,..\,n-1]) + \big[A[m] \neq B[n]\big] \end{array} \right\}}$$

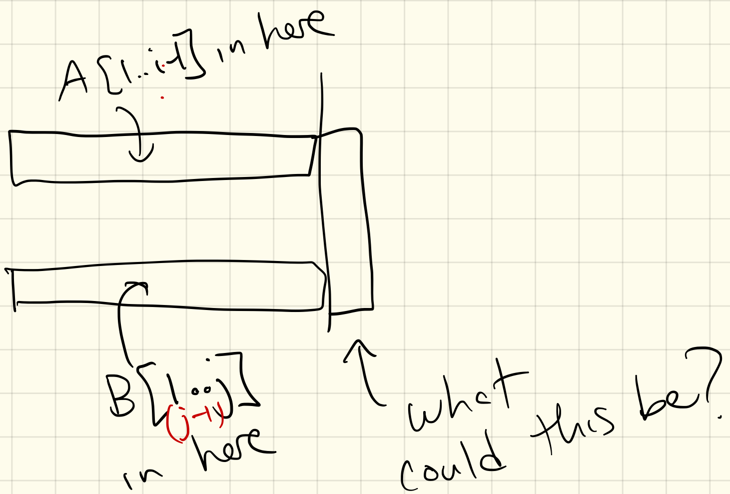
Turning this into a proper recursion:

Let $EDIT(i,j) := $ edit distance between:

$A[1..i]$

$B[1..j]$

$$Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{Bmatrix} Edit(i-1,j)+1, \\ Edit(i,j-1)+1, \\ Edit(i-1,j-1)+[A[i] \neq B[j]] \end{Bmatrix} & \text{otherwise} \end{cases}$$

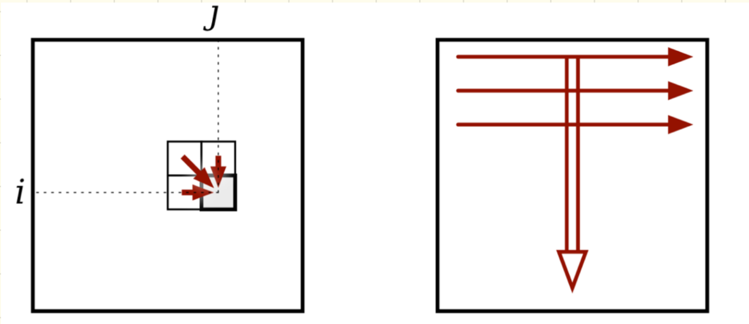


Now, don't bother analyzing the recursion.

(It's awful!)

Instead, be smart: memoize!

Table:



Algorithm:

```
EditDistance(A[1..m], B[1..n]):
  for j ← 1 to n
      Edit[0, j] ← j

  for i ← 1 to m
      Edit[i, 0] ← i
      for j ← 1 to n
          if A[i] = B[j]
              Edit[i, j] ← min {Edit[i − 1, j] + 1, Edit[i, j − 1] + 1, Edit[i − 1, j − 1]}
          else
              Edit[i, j] ← min {Edit[i − 1, j] + 1, Edit[i, j − 1] + 1, Edit[i − 1, j − 1] + 1}

  return Edit[m, n]
```

Example:

		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
A	1	**0**	1	2	3	4	5	6	7	8
L	2	1	**0**	1	2	3	4	5	6	7
T	3	2	1	1	2	3	4	**4**	5	6
R	4	3	2	2	2	**2**	3	4	5	6
U	5	4	3	3	3	3	3	4	5	6
I	6	5	4	4	4	4	**3**	4	5	6
S	7	6	5	5	5	5	4	4	5	6
T	8	7	6	6	6	6	5	**4**	5	6
I	9	8	7	7	7	7	**6**	5	5	6
C	10	9	8	8	8	8	7	6	6	6

The memoization table for *Edit*(ALGORITHM, ALTRUISTIC)

A	L	G	O	R		I		T	H	M
A	L	T		R	U	I	S	T	I	C