Algorithms & Complexity, Spring '26

Dynamic Programming

Recap

- Switch in office hours
- HW0 Solutions are late
  - ↳ apologies!

Last time: Longest Increasing subsequence

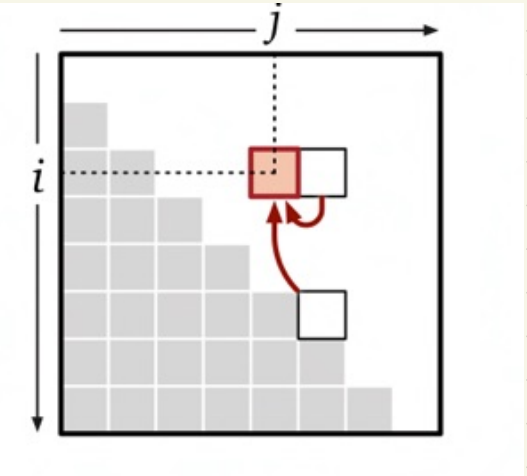
At any index j: can take (maybe) or skip:

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

recursive,

✓ Then, turn into loop to start at base case(s):

```
FastLIS(A[1..n]):
  A[0] ← −∞                          ⟨⟨Add a sentinel⟩⟩
  for i ← 0 to n                     ⟨⟨Base cases⟩⟩
      LISbigger[i, n+1] ← 0
  for j ← n down to 1
      for i ← 0 to j − 1             ⟨⟨...or whatever⟩⟩
          keep ← 1 + LISbigger[j, j+1]
          skip ← LISbigger[i, j+1]
          if A[i] ≥ A[j]
              LISbigger[i, j] ← skip
          else
              LISbigger[i, j] ← max{keep, skip}
  return LISbigger[0, 1]
```



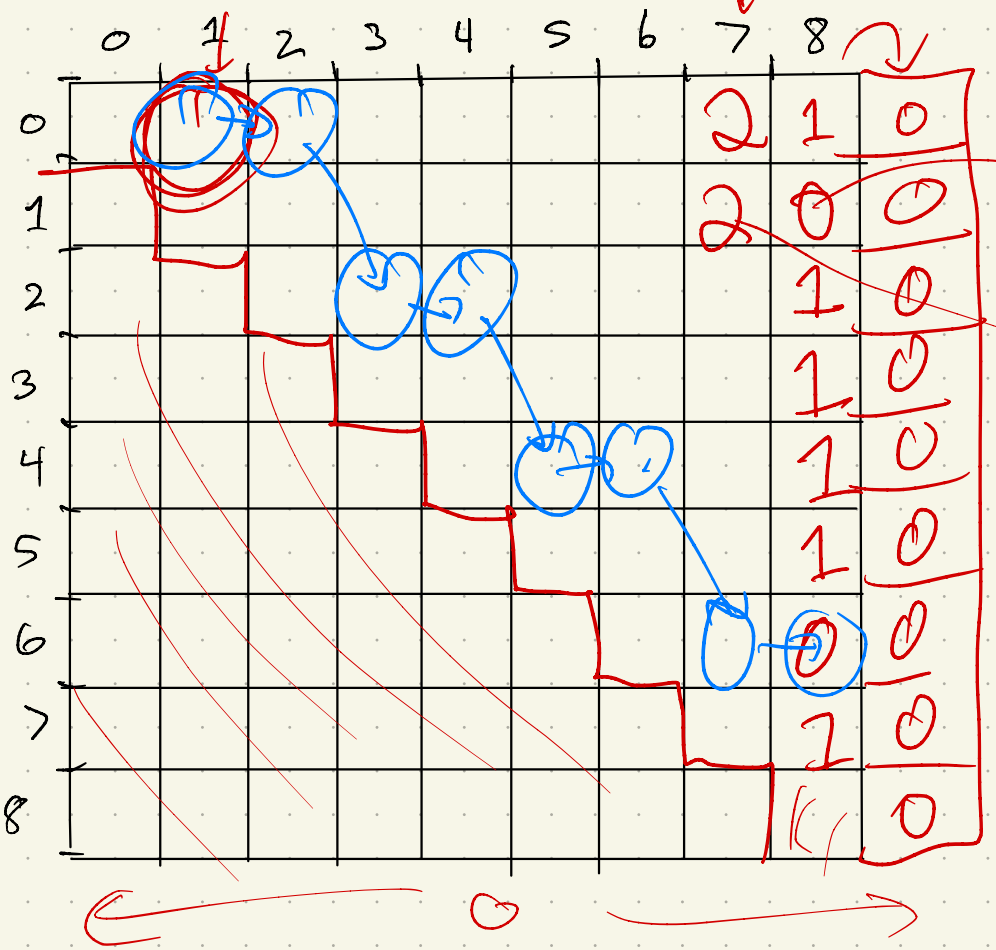
An example:

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

$A = [10, 2, 1, 4, 6, 11, 7, 9]$

0 1 2 3 4 5 6 7 8

reconstruct



LIS(0,7) = LIS(0,8) or LIS(7,8)

LIS(0,8)

LIS(1,8)

LIS(2,8)

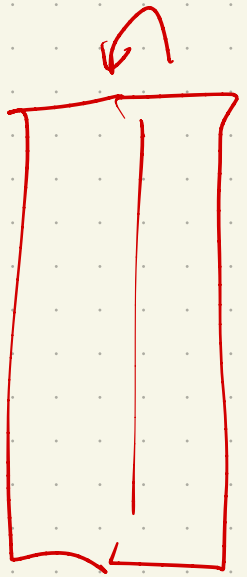
LIS(1,7)

LIS(1,8) 1+LIS(7,8)

Time: $O(n^2)$

Space: To get length: $O(n)$

keep 1 column



To get sequence

↳ no obvious $O(n)$ solution

Next one: Edit distance

HUGE in bioinformatics!

One of the basic tools in sequence alignment.

(I have a book with an entire chapter on how to optimize.)

Also: spell checkers, word prediction, etc.

From backtracking mindset: how to think recursively?

Consider 2 last characters:

A: A L G O R I T | M
1 _ _ _ _ _ _ | m

B: A L T R U I S T | C
1 _ _ _ _ _ _ | n

Options:

① Insert: add B[n] at end
1 + ↳ edit dist of A[1..m] + B[1..n-1]

② Delete: remove A[m]
1 + ↳ edit dist of A[1..(m-1)] + B[1..n]

③ Edit: free if A[m] = B[n]
↳ edit dist A[1..m-1] + B[1..n-1]
+ {1 or 0}

Example: TGCATAT to ATCCGAT

TGCATAT
↓ delete last T
TGCATA
↓ delete last A
TGCAT
↓ insert A at the front
ATGCAT
↓ substitute C for G in the third position
ATCCAT
↓ insert a G before the last A
ATCCGAT

TGCATAT
↓ insert A at the front
ATGCATAT
↓ delete T in the sixth position
ATGCAAT
↓ substitute G for A in the fifth position
ATGCGAT
↓ substitute C for G in the third position
ATCCGAT

recursively ↓

	T	G	C		A	T	A	T
A	T	C	C	G	A	T		

insert +1 +0 +0 delete +1 +0 +1 +0

recursively ↓

	T	G	C	A	T	A	T
A	T	C	C	G		A	T

+1 +1 +1 +1 ⇒ 4

11 5 cost

Input: $A[1..m]$ & $B[1..n]$

$Edit(i, j)$ = min cost edit seq from $A[1..i]$ & $B[1..j]$

$$= \min \begin{cases} \text{delete: } 1 + Edit(i-1, j) \\ \text{insert: } 1 + Edit(i, j-1) \\ \text{edit: } Edit[i-1, j-1] + [A[i] \neq B[j]] \end{cases}$$

Base case: if $i = 0$: $B[1..j]$ (A empty) $= j$

or $j = 0$: $A[1..i]$ (B empty) $= i$

So:

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

↳ why? +1 or +0 depending on match

Next:

What do we store in?

& how can we adapt loop?

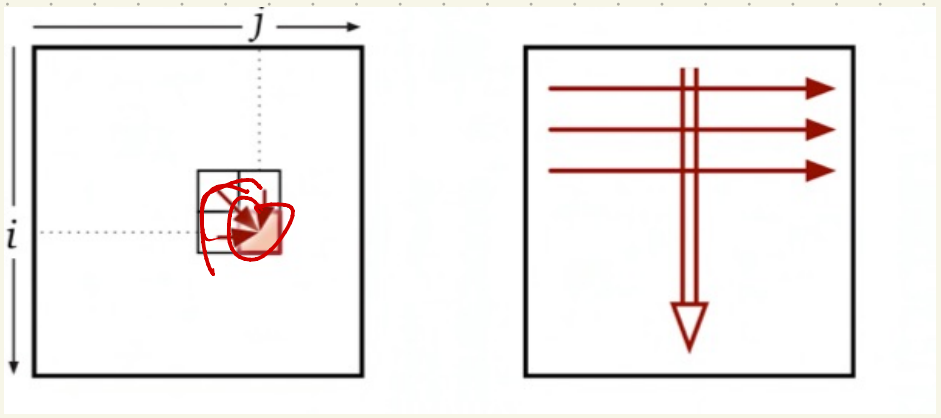
integer for each (i,j) with $1 \leq i \leq m$ & $1 \leq j \leq n$

use base case:

All in if $i=0$ or $j=0$

Final code:

```
EDITDISTANCE(A[1..m], B[1..n]):
  for j ← 0 to n
      Edit[0, j] ← j
  for i ← 1 to m
      Edit[i, 0] ← i
      for j ← 1 to n
          ins ← Edit[i, j − 1] + 1
          del ← Edit[i − 1, j] + 1
          if A[i] = B[j]
              rep ← Edit[i − 1, j − 1]
          else
              rep ← Edit[i − 1, j − 1] + 1
          Edit[i, j] ← min {ins, del, rep}
  return Edit[m, n]
```



Runtime: $O(mn)$

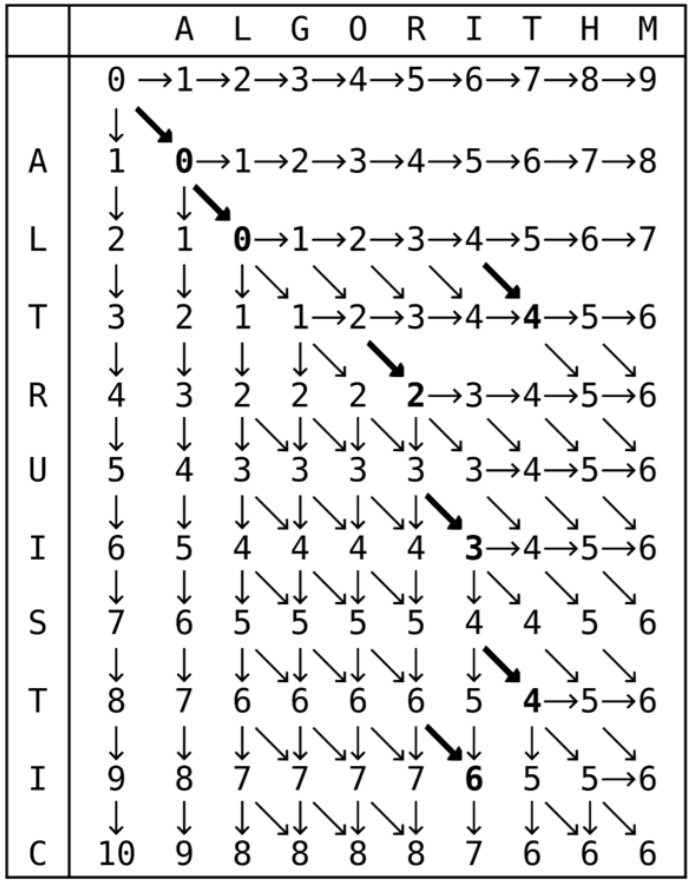
$O(1)$ time per-cell in 2d array

or $O((\max\{m,n\})^2)$

Space: $O(mn)$

Example

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

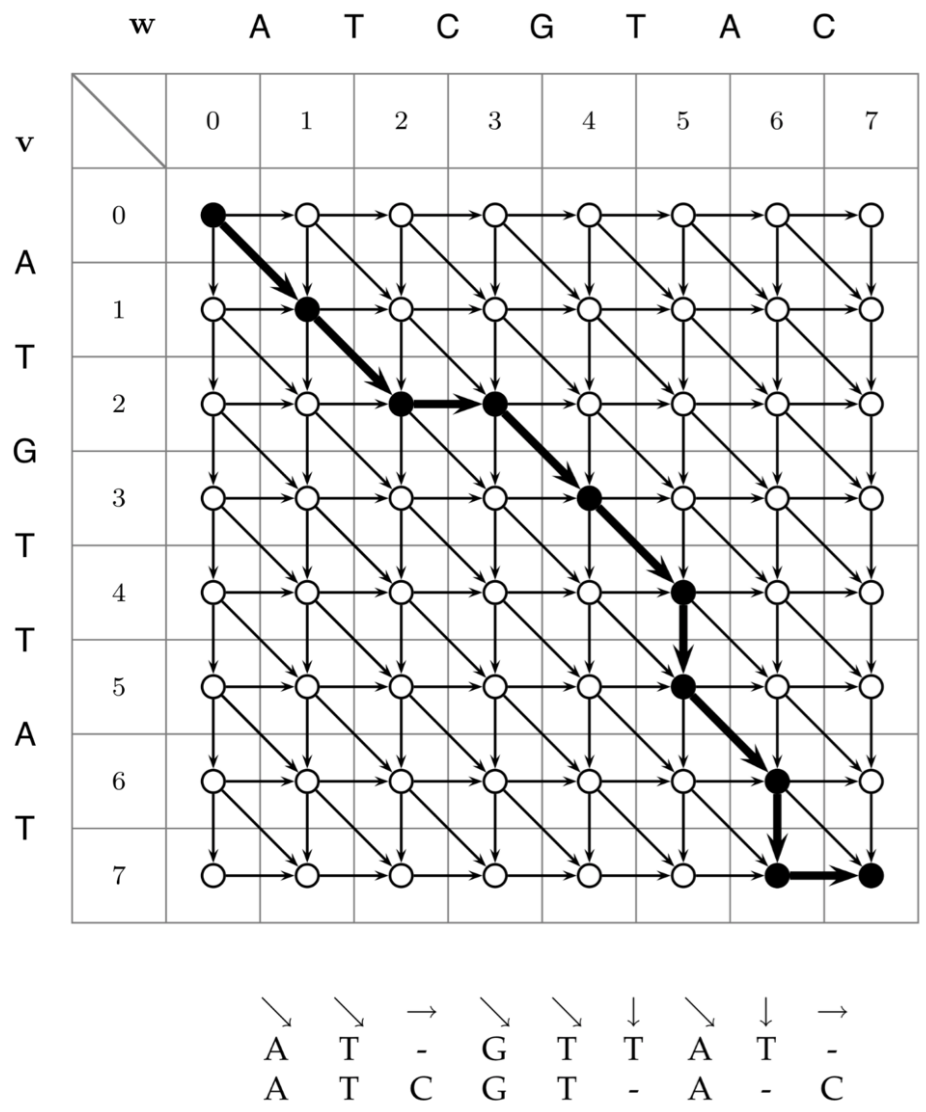


The memoization table for *Edit*(ALGORITHM, ALTRUISTIC)

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

Genetic sequence example:

		0	1	2	2	3	4	5	6	7	7
v	=		A	T	-	G	T	T	A	T	-
			|	|		|	|		|		
w	=		A	T	C	G	T	-	A	-	C
		0	1	2	3	4	5	5	6	6	7



Note: can think of underlying DAG here, & propogate in any topological sort order.

Not in book, but related: LCS

Longest Common Subsequence Problem:
Find the longest subsequence common to two strings.

Input: Two strings, **v** and **w**.

Output: The longest common subsequence of **v** and **w**.

Similar idea:

Let $S(i,j)$ = longest common subsequence of $V[1..i]$ & $W[1..j]$

Consider last letters:

① match: $1 + S(i-1, j-1)$, $S(i-1, j)$, $S(i, j-1)$

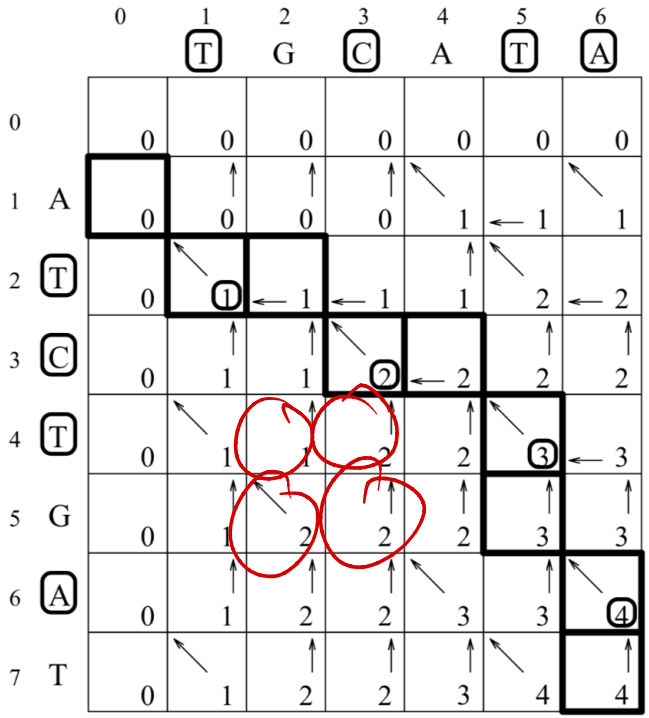
② not match: $\max \{ S(i-1, j), S(i, j-1) \}$

Backtracking

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, & \text{if } v_i = w_j \end{cases}$$

$s_{i,j} = S(i,j)$

and change to DP:



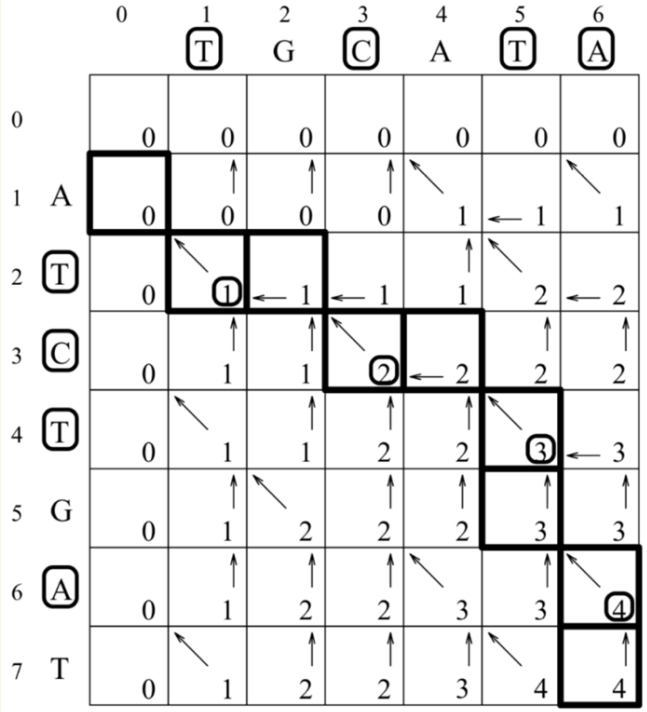
Computing similarity s(V,W)=4
V and W have a subsequence TCTA in common

Resulting alignment:

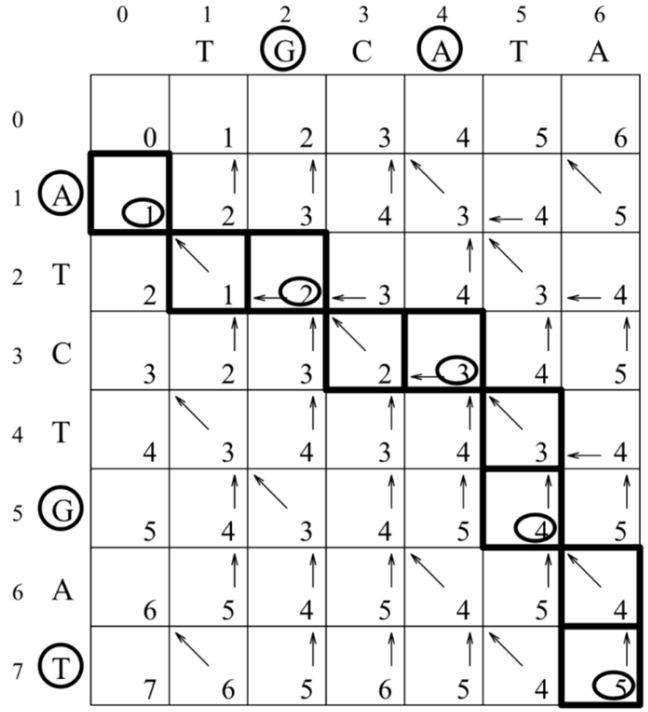
Alignment:

A T - C - T G A T
- T G C A T - A -

Note: Edit distance ≠ LCS
(but they are related)



Computing similarity s(V,W)=4
V and W have a subsequence TCTA in common



Computing distance d(V,W)=5
V can be transformed into W by deleting A,G,T and inserting G,A

Alignment:

A T - C - T G A T
- T G C A T - A -

A look at biology

LCS is a way to score similarity:

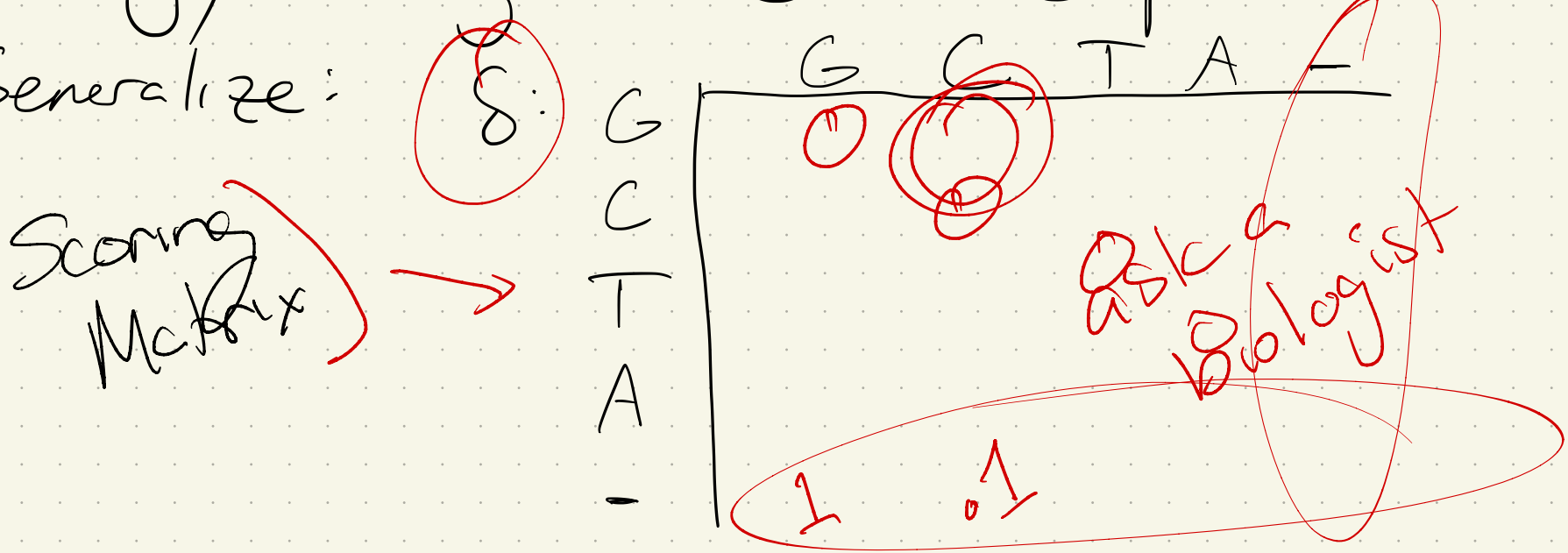
- +1 for a match
- +0 for a mismatch (indel)

Edit distance is too!

- insertion, deletion + substitution all cost +1.

Biology changes are more complex...

Generalize:



New goal:

Global Alignment Problem:
Find the best alignment between two strings under a given scoring matrix.

Input: Strings **v**, **w** and a scoring matrix δ.

Output: An alignment of **v** and **w** whose score (as defined by the matrix δ) is maximal among all possible alignments of **v** and **w**.

Same type of recursion:

When looking at v[1..i] and w[1..j]

① Align reuse $(v[1..i-1], w[1..j-1]) + \delta(v[i], w[j])$

② insert: $v[1..i], w[1..j-1] + \delta(w[j], -)$

③ del: $v[1..i-1], w[1..j] + \delta(\text{delete } v[i])$

Many more variants

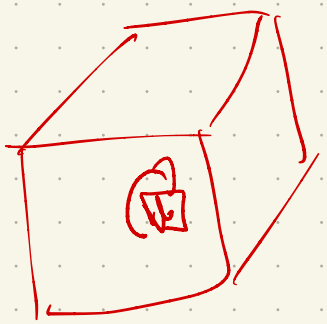
Local Alignment Problem:
Find the best local alignment between two strings.

Input: Strings **v** and **w** and a scoring matrix δ.

Output: Substrings of **v** and **w** whose global alignment, as defined by δ, is maximal among all global alignments of all substrings of **v** and **w**.

or multiple alignments:

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || |  | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
 ||||| |   X|||| |          || XXX|||  | ||||  |
-ATTGC-G--ATTCGTAT------GGGACA-TGGATGCATGCAG-TGAC
```



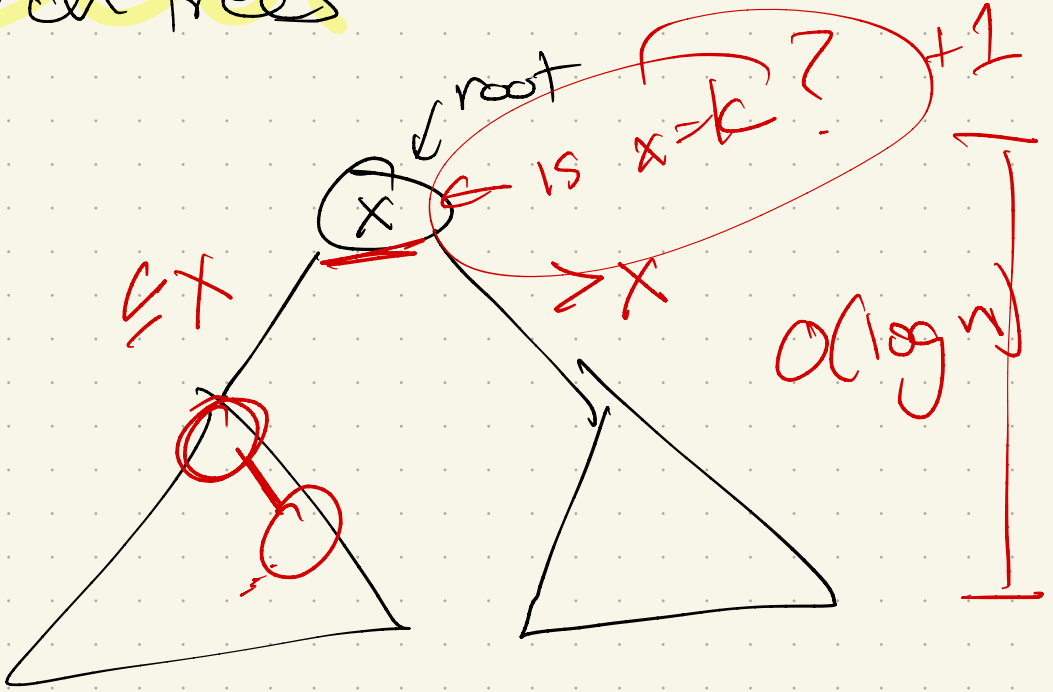
Core issues

How to get & scoring function?

↳ ask a biologist

Optimal Binary search trees

Recall: BSTs.



n nodes

Time to search for a value k in T:

O(depth of K in T)

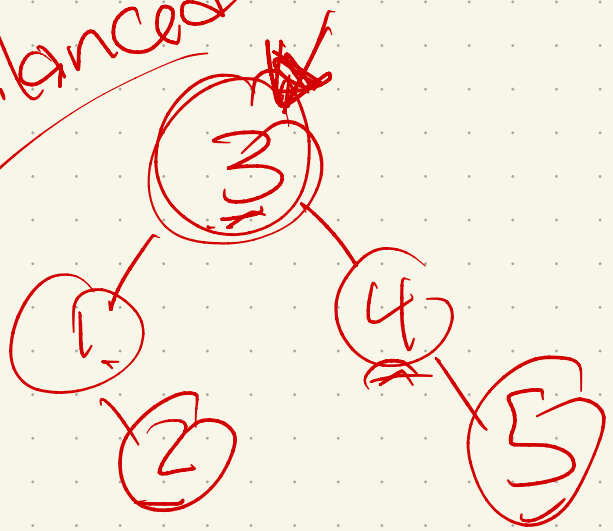
Goal: If I know how many times you'll look up each value in T_1 can I build the perfect BST?

Question: Why not balanced?

F	100	1	1	2	8
→ X	1	2	3	4	5

freq.
values

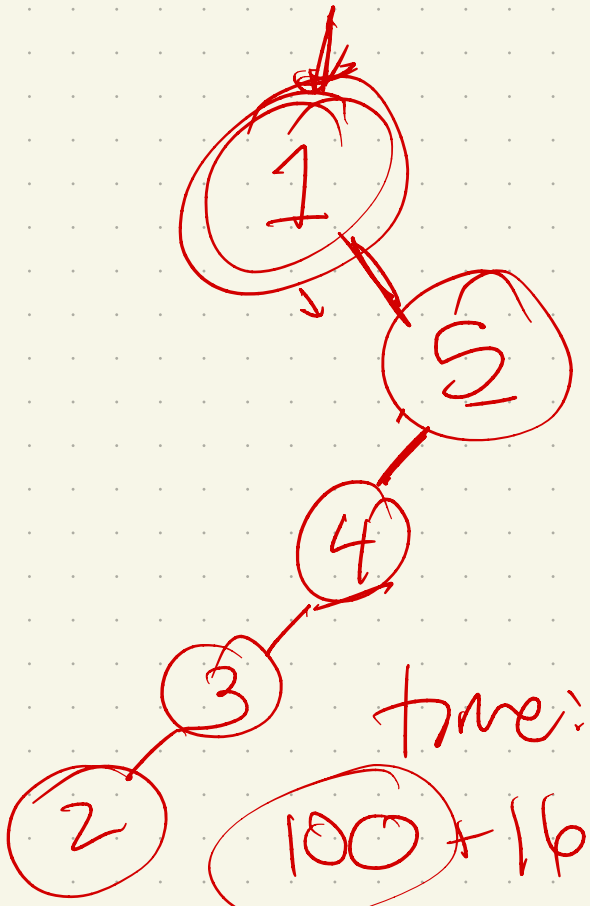
balanced



time:

200+3+1+4+24

= 232



time:

100+16+6+4+5

= 131 < 232

General problem: Given $X[1..n]$ & $F[1..n]$, where $X[i]$ has $F[i]$ searches, compute optimal BST:

$$\text{minimum cost} = \sum_i F[i] \cdot (\text{depth in } T)$$

Why?

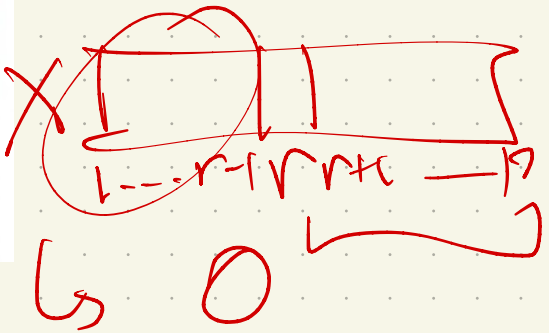
depth of $X[i]$ in T

Intuition: put max freq. on top

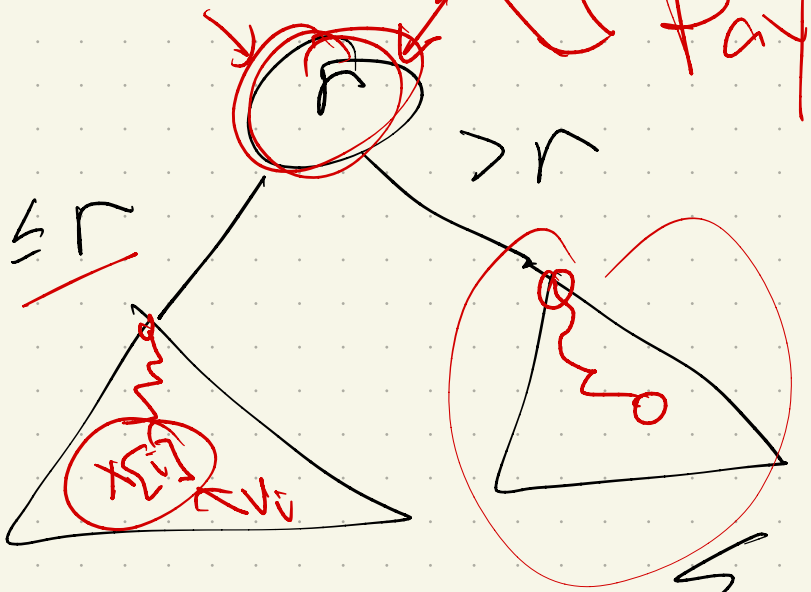
↳ GREEDY! (NO)

Last Chapter Assume X is sorted.

$$Cost(T, f[1..n]) = \sum_{i=1}^{n} f[i] + \sum_{i=1}^{r-1} f[i] \cdot \#\text{ancestors of } v_i \text{ in } left(T) + \sum_{i=r+1}^{n} f[i] \cdot \#\text{ancestors of } v_i \text{ in } right(T)$$



Why? Let root be r:



Pay for the root on every query

Essentially regrouping: $\sum_i F[i] \cdot depth = \sum_{levels\ k} (\text{frequencies of nodes at level} \geq k)$

Recursive (backtracking) strategy

$$Cost(T, f[1..n]) = \sum_{i=1}^{n} f[i] + \sum_{i=1}^{r-1} f[i] \cdot \#\text{ancestors of } v_i \text{ in } left(T) + \sum_{i=r+1}^{n} f[i] \cdot \#\text{ancestors of } v_i \text{ in } right(T)$$

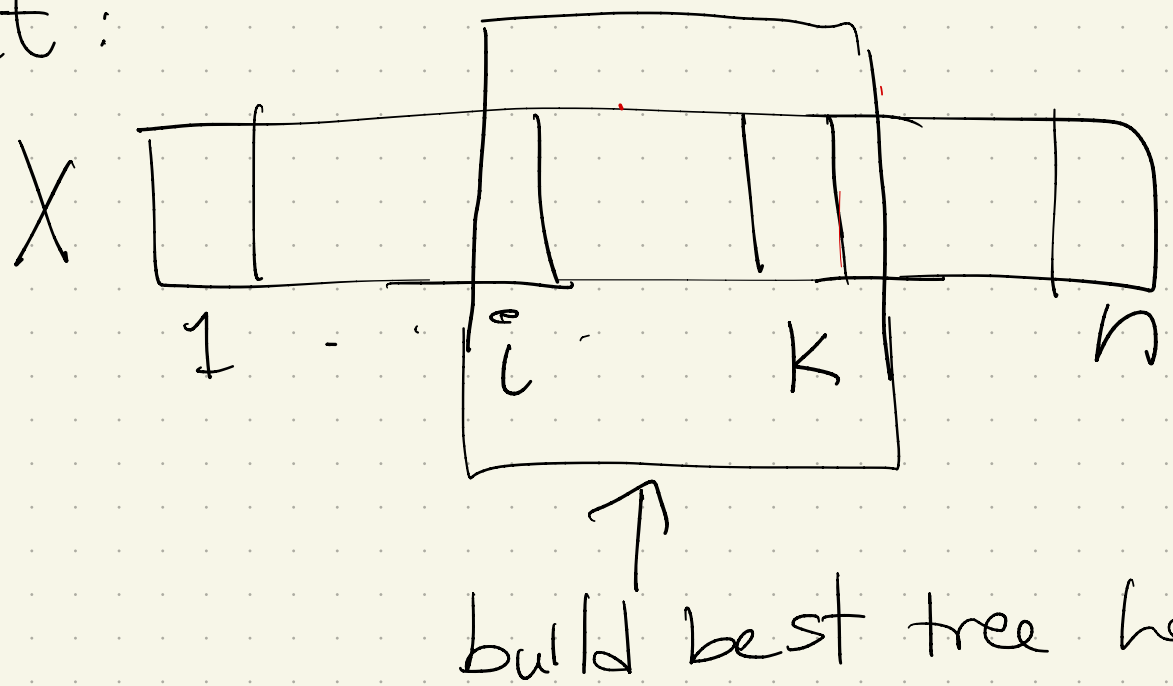
$$\Rightarrow \quad OptCost(i,k) = \begin{cases} 0 & \text{if } i > k \\ \sum_{j=i}^{k} f[i] + \min_{i \le r \le k} \left\{ \begin{array}{l} OptCost(i, r-1) \\ \quad + OptCost(r+1, k) \end{array} \right\} & \text{otherwise} \end{cases}$$

→ Choose best root!

How to memoize?

$$OptCost(i,k) = \begin{cases} 0 & \text{if } i > k \\ \sum_{j=i}^{k} f[i] + \min_{i \le r \le k} \left\{ \begin{matrix} OptCost(i, r-1) \\ + OptCost(r+1, k) \end{matrix} \right\} & \text{otherwise} \end{cases}$$

Remember input:



Everyone searches at root

↳ precompute

Let $F[i][k] = \sum_{j=i}^{k} f[j]$

Now:

$$OptCost(i,k) = \begin{cases} 0 & \text{if } i > k \\ \sum_{j=i}^{k} f[i] + \min_{i \le r \le k} \left\{ \begin{matrix} OptCost(i, r-1) \\ + OptCost(r+1, k) \end{matrix} \right\} & \text{otherwise} \end{cases}$$

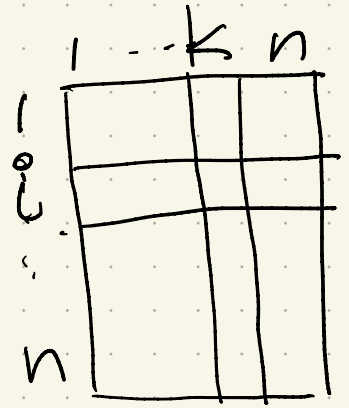
$\Rightarrow$ Opt Cost $(i,k) = \begin{cases} 0 \\ F[i][k] + \end{cases}$

Memoize: $0 \le i \le k \le n$

So: 2d table!

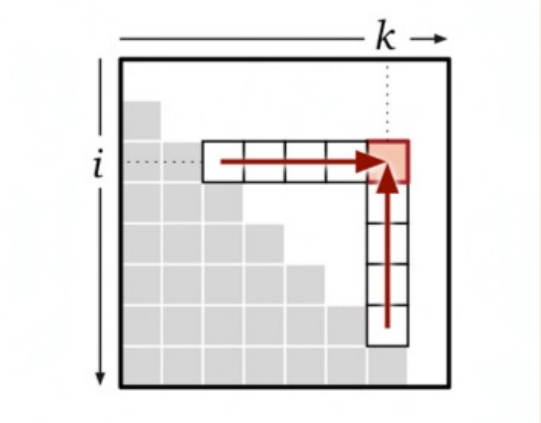
Each $O[i][k]$ needs:

- $F[i][k]$
- and



His picture (prettier):

$$OptCost(i,k) = \begin{cases} 0 & \text{if } i > k \\ F[i,k] + \min\limits_{i \le r \le k} \left\{ \begin{array}{l} OptCost(i, r-1) \\ \quad + OptCost(r+1, k) \end{array} \right\} & \text{otherwise} \end{cases}$$



So:

```
OptimalBST(f[1..n]):
  InitF(f[1..n])
  for i ← 1 to n+1
      OptCost[i, i−1] ← 0
  for d ← 0 to n−1
      for i ← 1 to n−d        ⟨⟨...or whatever⟩⟩
          ComputeOptCost(i, i+d)
  return OptCost[1, n]
```

Time:

Space:

Other ones in reading:

- Subset Sum: $O(nT)$

  but

- Independent Sets in trees:

  Not an array!

  For each node, need to store values

  ↳ Use the tree