

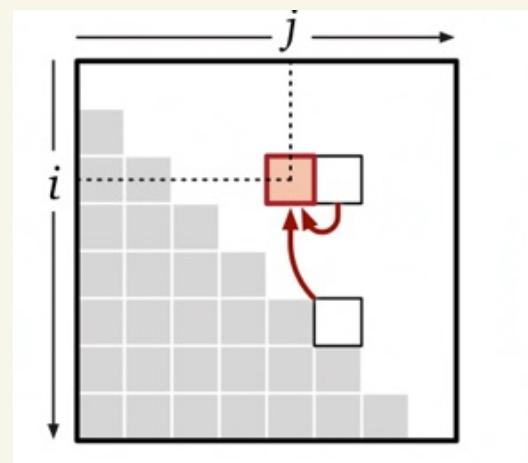

Last time: Longest Increasing Subsequence
At any index j : can take (maybe) or skip:

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

recursive,

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

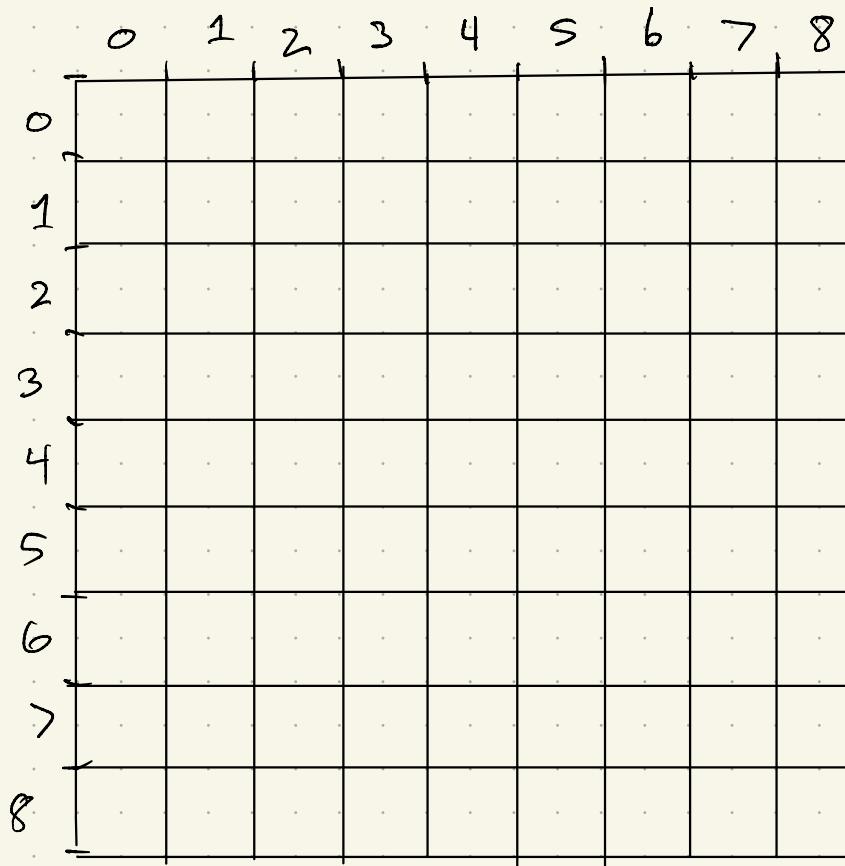
```
FASTLIS(A[1..n]):  
    A[0] ← -∞           ⟨⟨Add a sentinel⟩⟩  
    for i ← 0 to n      ⟨⟨Base cases⟩⟩  
        LISbigger[i, n + 1] ← 0  
    for j ← n down to 1  
        for i ← 0 to j - 1   ⟨⟨... or whatever⟩⟩  
            keep ← 1 + LISbigger[j, j + 1]  
            skip ← LISbigger[i, j +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



An example:

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

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



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 :

ALGORITHM

ALTRUISTIC

Options :

Example:

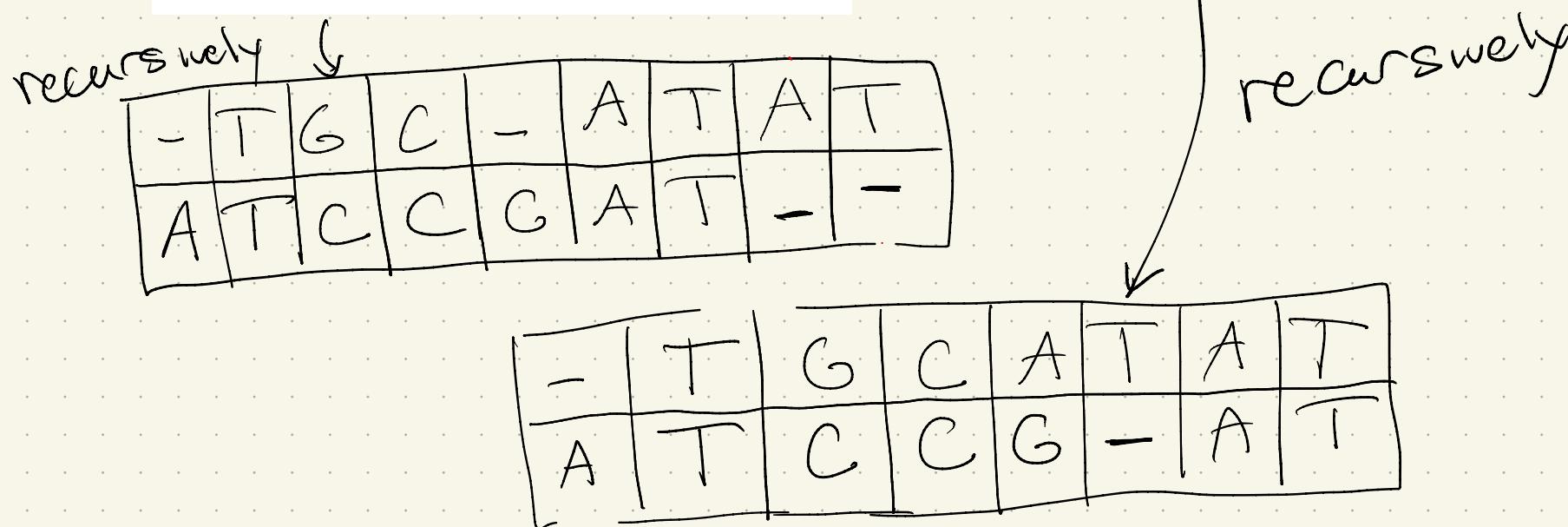
TGCATAT
to ATCCGAT

TGCATAT
 ↓
 TGCATA
 ↓
 TGCAT
 ↓
 ATGCAT
 ↓
 ATCCAT
 ↓
 ATCCGAT

delete last T
 delete last A
 insert A at the front
 substitute C for G in the third position
 insert a G before the last A

TGCATAT
 ↓
 ATGCATAT
 ↓
 ATGCAAT
 ↓
 ATGCCAT
 ↓
 ATCCGAT

insert A at the front
 delete T in the sixth position
 substitute G for A in the fifth position
 substitute C for G in the third position



Input: $A[1..m] \times B[1..n]$

$\text{Edit}(i, j)$

$\geq \min$

delete:

insert:

edit:

Base case: if $i=0$:

or $j=0$

So

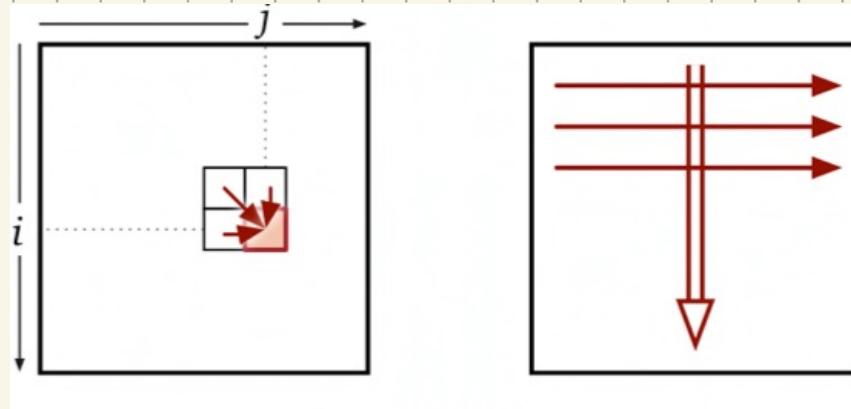
$$Edit(i, j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} Edit(i, j - 1) + 1 \\ Edit(i - 1, j) + 1 \\ Edit(i - 1, j - 1) + [A[i] \neq B[j]] \end{array} \right\} & \text{otherwise} \end{cases}$$

What do we store in?

& how can we adopt loop?

Final Code :

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



Runtime :

Space :

Example

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

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

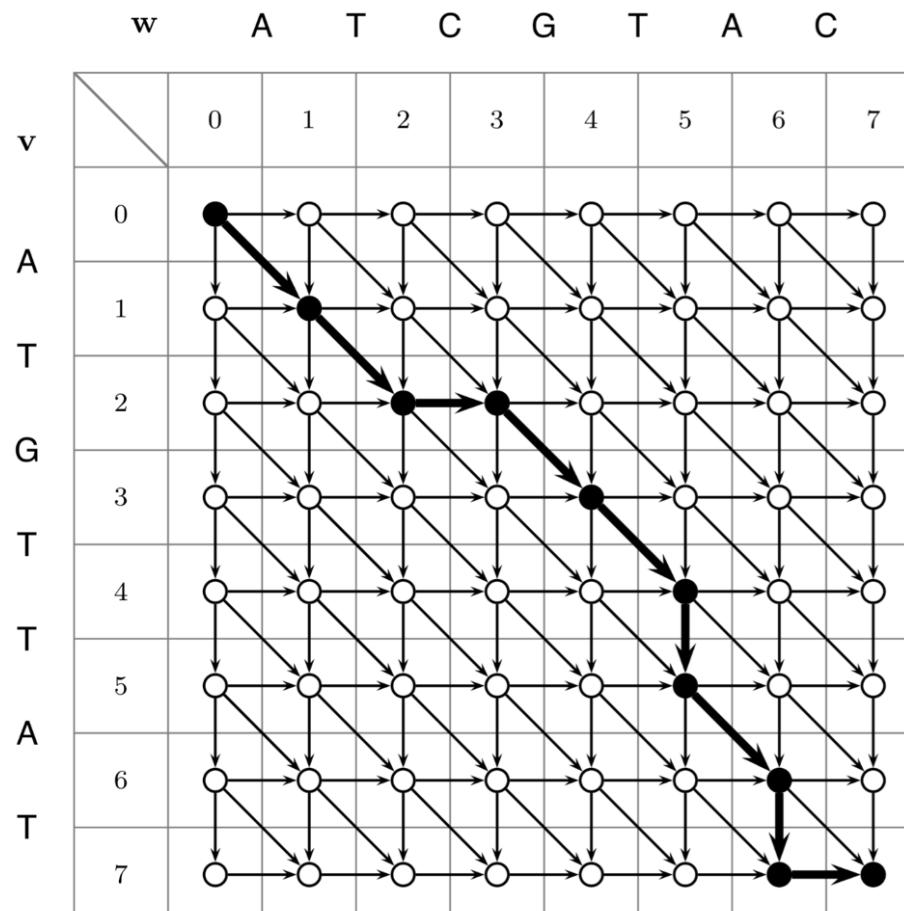
The memoization table for $Edit(\text{ALGORITHM}, \text{ALTRUISTIC})$

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

Genetic Sequence example!

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

	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



\searrow	\searrow	\rightarrow	\searrow	\searrow	\downarrow	\searrow	\downarrow	\rightarrow
A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

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

②

Backtracking

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

and change to DP.

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	A	0	0	0	1	-1	1
2	T	0	1	-1	1	2	-2
3	C	0	1	1	2	2	2
4	T	0	1	1	2	3	-3
5	G	0	1	2	2	3	3
6	A	0	1	2	2	3	4
7	T	0	1	2	2	3	4

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

Resulting alignments

Alignment: A T - C - T G A T
 - T G C A T - A -

Note: Edit distance \neq LCS
 (but they are related)

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1 A	0	0	0	0	1	1	1
2 T	0	①	1	1	1	2	2
3 C	0	1	1	②	2	2	2
4 T	0	1	1	2	2	③	3
5 G	0	1	2	2	2	3	3
6 A	0	1	2	2	3	3	④
7 T	0	1	2	2	3	4	4

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

	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1 A	①	2	3	4	3	4	5
2 T	2	1	②	3	4	3	4
3 C	3	2	3	③	4	5	
4 T	4	3	4	3	4	3	4
5 G	5	4	3	4	5	④	5
6 A	6	5	4	5	4	5	4
7 T	7	6	5	6	5	4	⑤

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: S: G C T A -

Scoring
Matrix

G
C
T
A
-

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

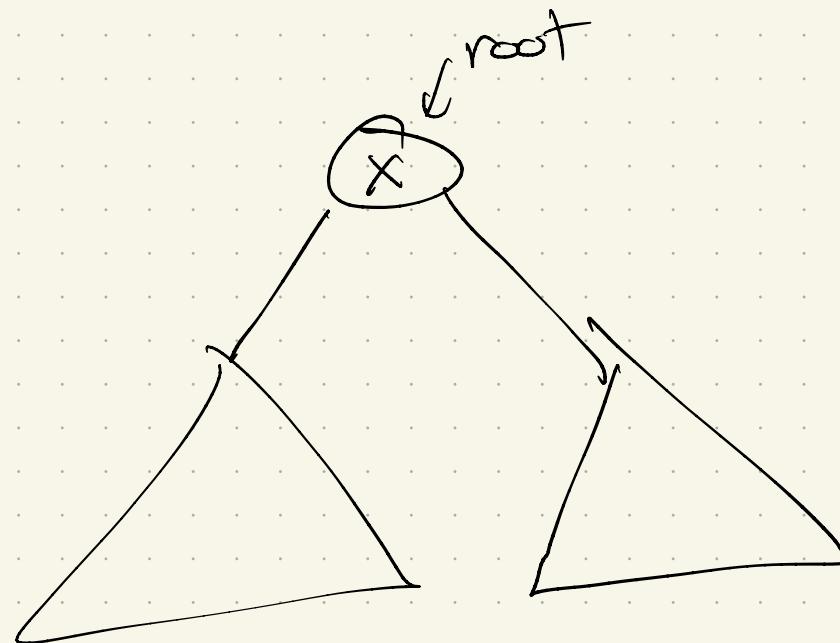
②

③

Many more variants

Optimal Binary Search trees

Recall: BSTs.



Time to search for
a value k in T :

Goal: If I know how many times you'll
look up each value in T , can I build the
perfect BST?

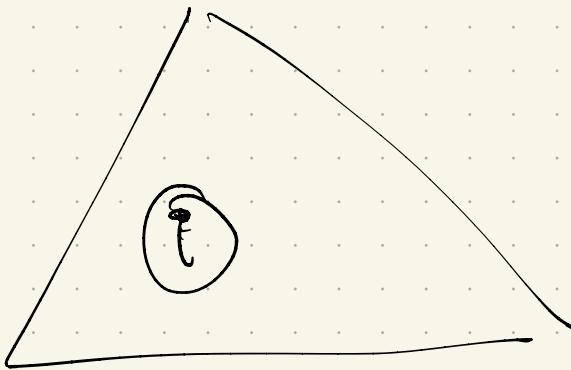
Question: Why not balanced?

F	100	1	1	2	8
X	1	2	3	4	5

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?

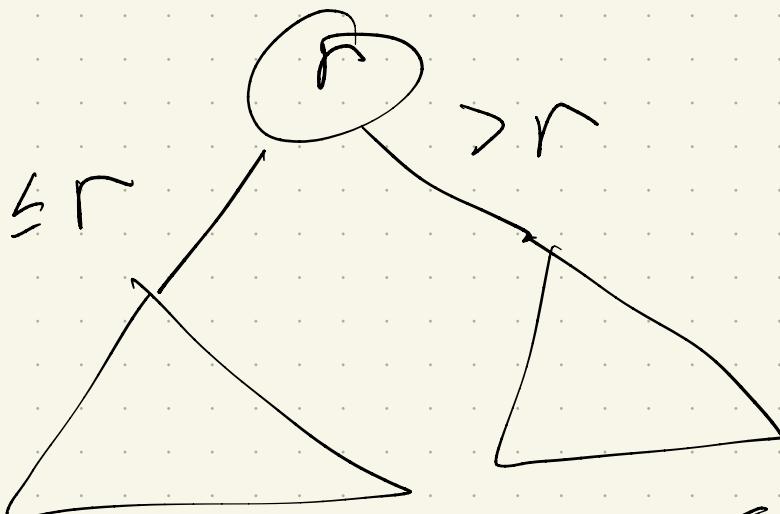


Intuition:

Last Chapter Assume X is sorted.

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

Why? Let root be r :



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

Recursive (Backtracking) Strategy

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



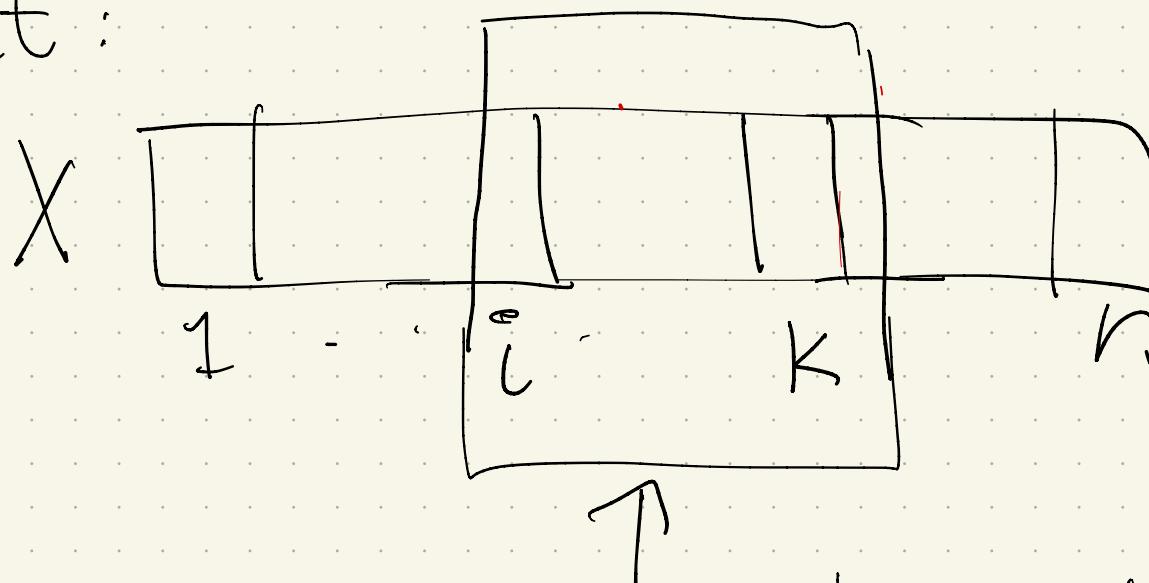
$$\text{OptCost}(i, k) = \begin{cases} 0 & \text{if } i > k \\ \sum_{j=i}^k f[j] + \min_{i \leq r \leq k} \left\{ \text{OptCost}(i, r-1) + \text{OptCost}(r+1, k) \right\} & \text{otherwise} \end{cases}$$

Choose best root!

How to memoize?

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

Remember Input:



Everyone searches at root
↳ precompute

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

Now:

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

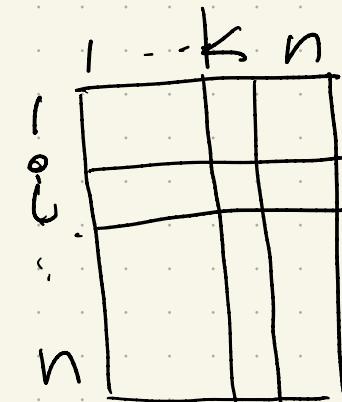
$$\Rightarrow OptCost(i, k) = \left\{ \begin{array}{l} 0 \\ F[i][k] + \end{array} \right.$$

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

So: 2D table!

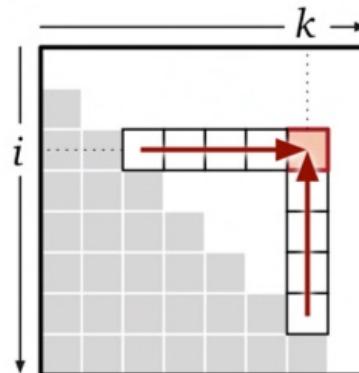
Each $O[i][k]$ needs:

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



Hs picture (prether):

$$OptCost(i, k) = \begin{cases} 0 & \text{if } i > k \\ F[i, k] + \min_{i \leq r \leq k} \left\{ OptCost(i, r - 1) + OptCost(r + 1, k) \right\} & \text{otherwise} \end{cases}$$



So:

```
OPTIMALBST( $f[1..n]$ ):  
    INITF( $f[1..n]$ )  
    for  $i \leftarrow 1$  to  $n + 1$   
         $OptCost[i, i - 1] \leftarrow 0$   
    for  $d \leftarrow 0$  to  $n - 1$   
        for  $i \leftarrow 1$  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