More dynamic programming

Tuesday, October 13, 2020 1:31 PM

Longest increasing subsequence

e.g. $A = [6, 3, 5, 2, 7, 8, 1]$

subsequence: e.g. 6, 5, 2

increasing subsequence... e.g. [6, 7, 8]

[3, 5, 7, 8] ? ✓

[2, 7, 8] ?

names of subproblems.

$LIS(i, seqSoFar)$ = longest subsequence of $A[i \ldots n]$ that is increasing if appended to seqSoFar.

0 if $i = n+1$

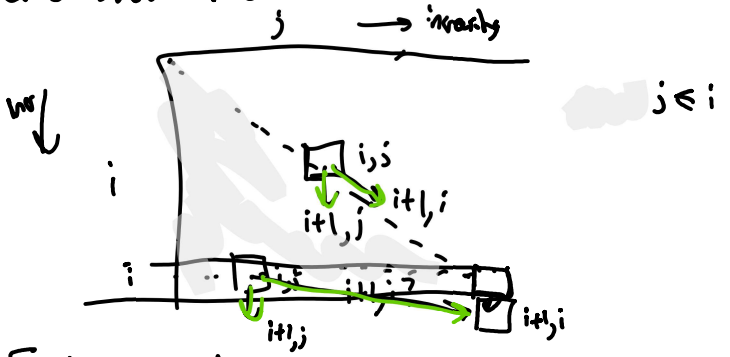
$\max \begin{cases} 1 + LIS(i+1, seqSoFar + +A[i]) & \text{if } A[i] \geq \text{last element of seqSoFar} \quad \text{// include } A[i] \\ LIS(i+1, seqSoFar) & \text{// exclude } A[i] \end{cases}$

$LIS(i, j)$ = length of longest subsequence of $A[i, \ldots n]$ containing elements larger than $A[j]$

0 if $i = n+1$

$\max \begin{cases} 1 + LIS(i+1, i) & \text{if } A[i] \geq A[j] \quad \text{// include } A[i] \\ LIS(i+1, j) & \text{// exclude } A[i] \end{cases}$

Memo data structure



Evaluation order:

each subproblem i,j depends on $i+1,j$ $i+1,i$

so evaluate in order of: decreasing i.

or decreasing j.

row major decreasing i, decreasing j

col major in decreasing i, decreasing j.

Estimate running time:
subproblems. $O(n^2)$

Time for each subproblem: $O(1)$

Running time: $O(n^2)$

Edit Distance.

- Distance between strings
- min # of edit operations needed to turn string X into string Y

edit operation: - insertion of 1 character
- delete 1 character
- substitute 1 character

ex of operations

sub CAT → CAB subst T for B

ins CAT → CART

del CAT → CA

ex

A L G O R I T H M

A L T R U I S T I C

. . 1 2 3 . 4 5 6 7

Gaps in?

A L G O R I T H M

A L G R I T H M - on the top means insert

X= A L G O R — I — T H M 6!

A L T — R U I S T I C

1 2 3 4 5 6

- on the bottom means: delete

See course notes for a proof of the following.

First recurrence:

Dist(X, Y) edit distance X to Y.

Base — Dist(ε, Y) = |Y| // |Y| insertions

base case — $Dist(X, \epsilon) = |X|$ // |X| deletions.

inductive case — $Dist(aX', bY') =$

$$\min \begin{cases} 1 + Dist(aX', Y') & \text{// inserting} \\ 1 + Dist(X', bY') & \text{// deleting} \\ (a \neq b) + Dist(X', Y') & \text{// substituting} \end{cases}$$

($(a \neq b)$ is 1 if $a \neq b$, 0 otherwise)

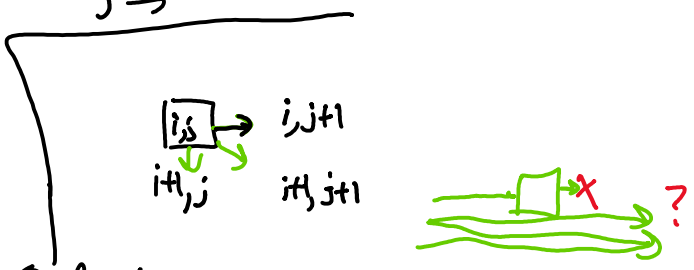
subproblem: $\frac{b}{b}\frac{aX'}{Y'}$ start, $\frac{a}{b}\frac{X'}{bY'}$, $\frac{a}{b}\frac{X'}{Y'}$

$Dist(i, j)$ (1 indexed strings) — edit distance from $X[i \ldots m]$ to $Y[j \ldots n]$; $|X| = m$, $|Y| = n$; $|Y[j \ldots n]| = n - j + 1$

- $Dist(m+1, j) = n - j + 1$ // $n-j+1$ insertions

$Dist(i, n+1) = m - i + 1$ // $m-i+1$ deletions.

$$Dist(i, j) = \min \begin{cases} 1 + Dist(i, j+1) & \text{// insertion} \\ 1 + Dist(i+1, j) & \text{// deletion} \\ (X[i] \neq Y[j]) + Dist(i+1, j+1) & \text{// subst} \end{cases}$$



Evaluation Order:
decreasing i and decreasing j.

e.g. row major w/ decreasing i and decreasing j.

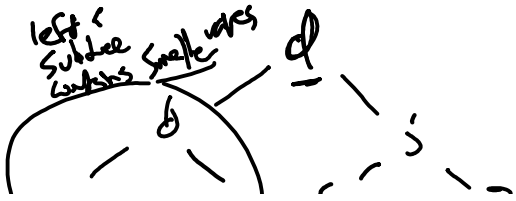
# of subproblems: $O(nm)$

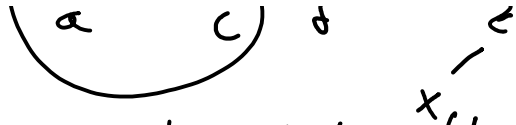
Cost of each subproblem: $O(1)$

Overall: $O(nm)$ running time.

Pseudocode: ... leave to later if time.

Optimal Binary Search Tree





In balanced tree: depth $O(\log_2 n)$ worst case.

```
Search(X, T):
    if T = ∈ then False
    if T.val = X then True
    if X < T.val then Search(X, T.left)
    else:             Search(X, T.right).
```

What about weighted average case search time?

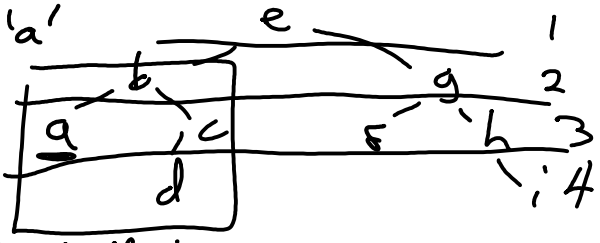
Input: $A = [a, b, c, d, e, f, g, h, i]$ (sorted)

Frequency $= [.5, .1, .1, .01, .01, .2, .01, .01, .01]$

Avg Cost:

$1 \cdot (.01)$
$+ 2 \cdot (.1 + .01)$
$+ 3 \cdot (.5 + .1 + .2 + .01)$
$\ldots$

half of all queries are for 'a'



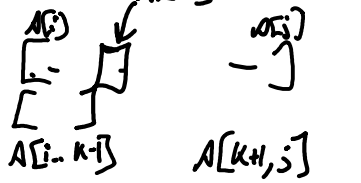
What's the tree that gives best avg cost?

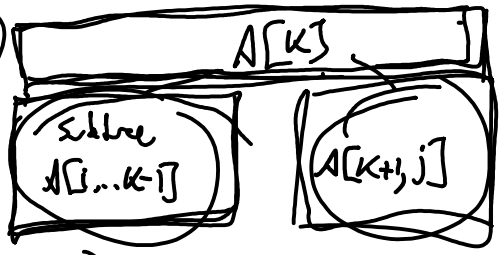
optBST(i, j) — cost of best tree for interval $A[i \ldots j]$.

if i = j: $1 \cdot F[i]$

else:

$$\min_{i \le k \le j} \begin{cases} \sum F[i \ldots j] \\ + \text{optBST}(i, k-1) \\ + \text{optBST}(k+1, j) \end{cases}$$





optBST(A, F):

Let Opt be a $(n+1) \times (n+1)$ array.

Let Root be a $(n+1) \times (n+1)$ array.

For ... // base cases.

For $i = n$ down to 1:

  For $j = i$ up to n

    Best := ∞

    BestK := ∞ ← what value of k gave me the best score?

```
for k = i upto j:
    X = Σ Freq[i...j] + opt[i,k-1] + opt[k+1, j]
    if X < Best:
        BestK := k
        Best := X
Opt[i,j] := Best
Root[i,j] := BestK
return opt[1,n]
```

To read back the structure:

1. — Store the value of k that gave the best
2. — Recursive function to read it back:

```
BST(i,j):                    // make use of OptBST & Roots
    if i=j then return T(A[i], ε, ε)
    else return T(A[Root[i,j]], BST(i, Root[i,j]-1),
                                BST(Root[i,j]+1, j))
```

BST is defined inductively as:

- ε is the empty tree
- T(val, t_L, t_R) is a node with value val, left subtree t_L, right subtree t_R