Start: warm-up problem 1

CSCI 570 Summer 2016
Dynamic Programming II

"Cost" of words on a line:

Cost (i, j)

if words $i..j$ don't fit, return ∞

if $j = n$: return 0

else return $\left(M - j + i - \sum_{k=i}^{j} l_k\right)^3$

// tot_len[i,j] = $\sum_{k=i}^{j} \ell_k$

```
for i = 1 to n
    tot_len[i,i] = ℓ_i
    cost[i,i] = ----
    for j = i+1 to n
        tot_len[i,j] = tot_len[i,j-1] + ℓ_j
        if M - i + j - tot_len[i,j] < 0
            cost[i,j] = ∞
        else if j == n
            cost[i,j] = 0
        else
            cost[i,j] = (M - i + j - tot_len[i,j])^3
```

total: $\Theta(n^2)$

$\Theta(1)$ each

Printing Neatly: Recursive Solution

def OPT(j): optimal cost to print words 1...j

if j = 0: return 0

// pretend $1 \le k \le j$ is 1st word

// on same line as j.

// then OPT(j) = OPT(k-1) + cost(k, j)

return $\min_{1 \le k \le j} \{ OPT(k-1) + cost(k, j) \}$

When iterative, n elements to fill, O(n) each $O(n^2)$

O(n) each

Printing Neatly: Iterative Solution

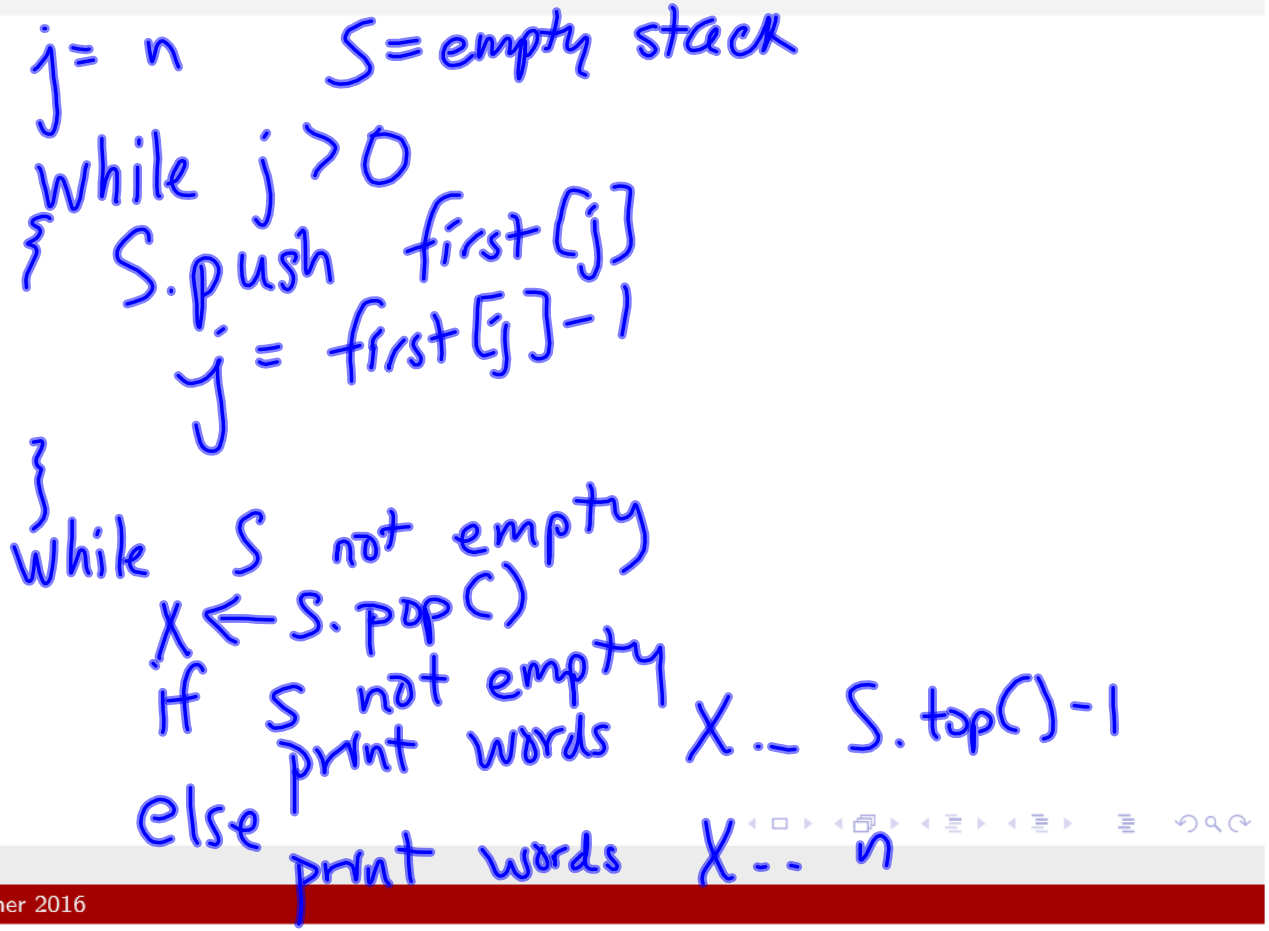
- precompute all cost[i,j]
- declare OPT[0..n]

```
OPT[0] = 0
for j = 1 to n
    OPT[j] = OPT[j-1] + cost[j,j]
    first[j] = j
    for k = j-1 down to
        costk = OPT[k-1] + cost[k,j]
        if costk < OPT[j]
            OPT[j] = costk
            first[j] = k
```

OK to break loop when costk = ∞

first[j] means "if j last word on a line, first[j] should be first on that line"

Printing Neatly: Produce the paragraph



Knapsack: Recursive Solution

OPT(i, x): "optimal value, items 1..i, using $x \le W$ weight"

O(1) if OPT(·) is array lookup.

How many subproblems? $\Theta(nW)$

So $\Theta(nW) \cdot O(1)$ → $O(nW)$

```
if i < 1 : return 0
or x < 1
if w_i > x   return OPT(i-1, x)
return max(v_i + OPT(i-1, x-w_i), OPT(i-1, x))
// take ith item or don't
```

Knapsack: Iterative Solution

declare OPT[0...n, 0...W]

for x = 0 to W : OPT[0, x] = 0

for i = 0 to n OPT[i, 0] = 0

for i = 1 to n

for x = 1 to W

if w[i] > x OPT[i, x] = OPT[i-1, x]

else

OPT[i, x] = Max - - - -

Knapsack: Produce the set

```
i = n,  x = W
while i > 0 :  // decide to take i^th?
    if OPT[i,x] == OPT[i-1,x]
        i = i-1
    else
        Output i^th
        x = x - w[i]
        i--
```

LCS: Recursive Solution

Goal: LCS[|X|, |Y|]

LCS(i,j): longest common subseq of X[1..i] and Y[1..j]

```
if i<1 or j<1
    return 0
else if X[i]==Y[j]
    return LCS(i-1,j-1)+1
else
    return max(LCS(i-1,j), LCS(i,j-1))
    // del X[i]    del Y[j]
```

LCS: Iterative Solution

declare $LCS[0..m, 0..n]$

$m = |X|$ $n = |Y|$

for $i = 0$ to m

for $j = 1$ to n

$LCS[i,0] = 0$

$LCS[0,j] = 0$

for $i = 1$ to m

for $j = 1$ to n

if $X[i] == Y[j]$

$LCS[i,j] = 1 + LCS[i-1, j-1]$

else

$LCS[i,j] = \max \ldots$

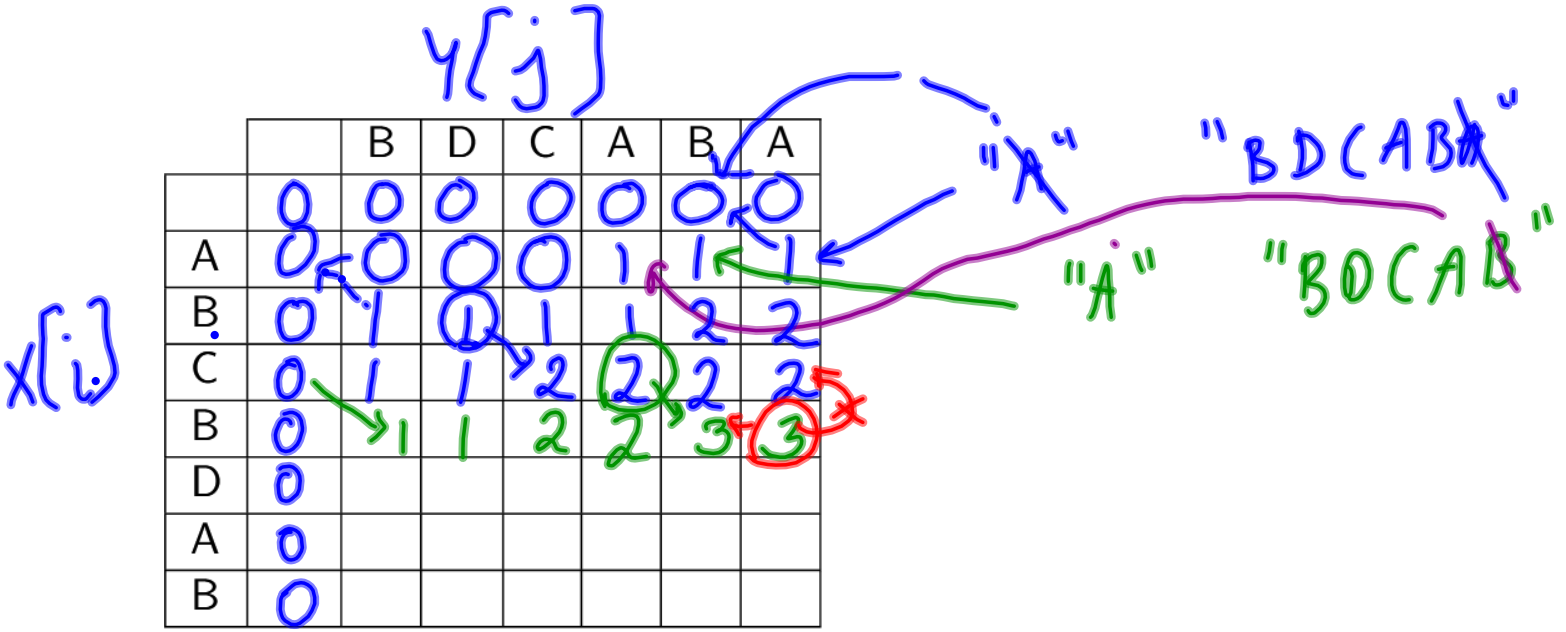
$O(mn)$ iterations

$O(1)$ time each

$O(mn)$

$O(1)$

LCS Illustration

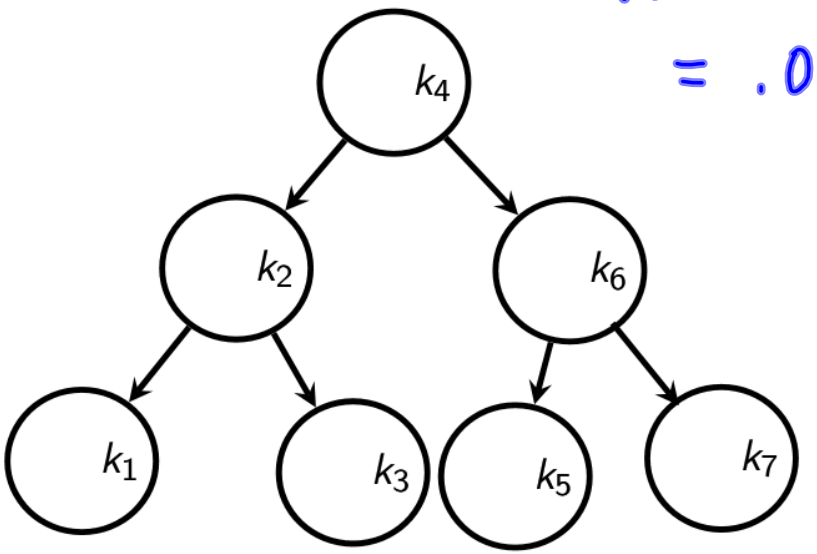


		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1	2	2	2	2
B	0	1	1	2	2	3	3
D	0						
A	0						
B	0						

LCS: Produce Output

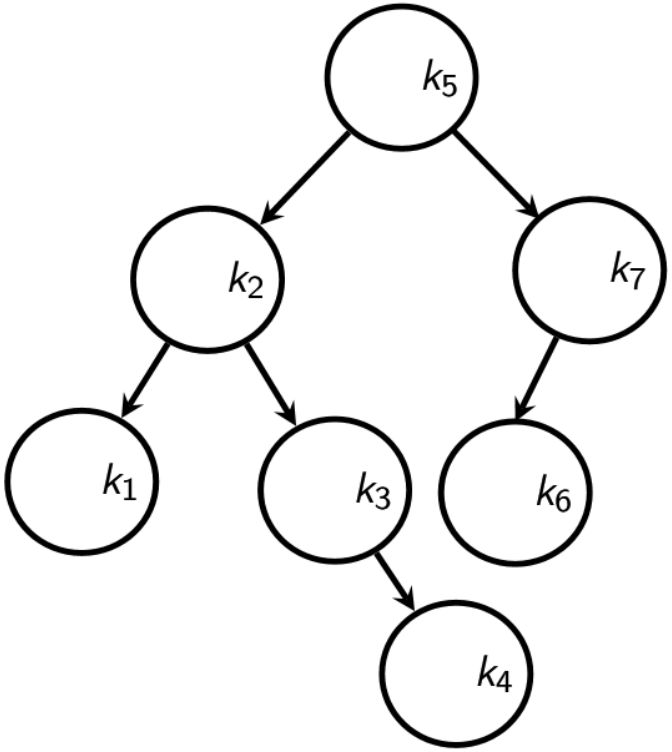
```
i = |X|, j = |Y|
while i > 0 and j > 0:
    if X[i] == Y[j]
        push X[i]
        i--
        j--
    elif LCS[i-1,j] > LCS[i,j-1]
        i--
    else
        j--
```

First Tree



$.01(1) + .29(2) + .7(3)$
$= .01 + .58 + 2.1$
$= 2.69$

Second Tree



expected lookup cost:

$\sum_{i \in T} d_i \cdot p_i$

where d_i = depth of node i.

$1(.22)$ $.22$

$+ 2(.45)$ $+.9$ $\Big\} 1.12$

$+ 3(.32)$ $+.96$

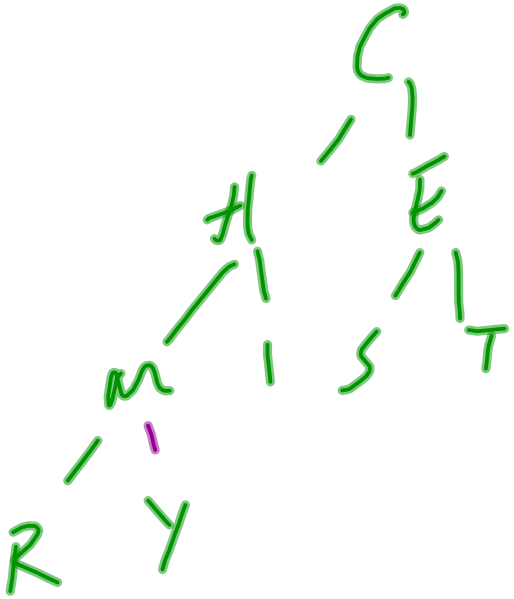
$+ 4(.01)$ $+.04$ $\Big\} 1$

2.12

Recursive Solution

Iterative Solution

Produce the tree


C
H
E
m
S
T
R
Y