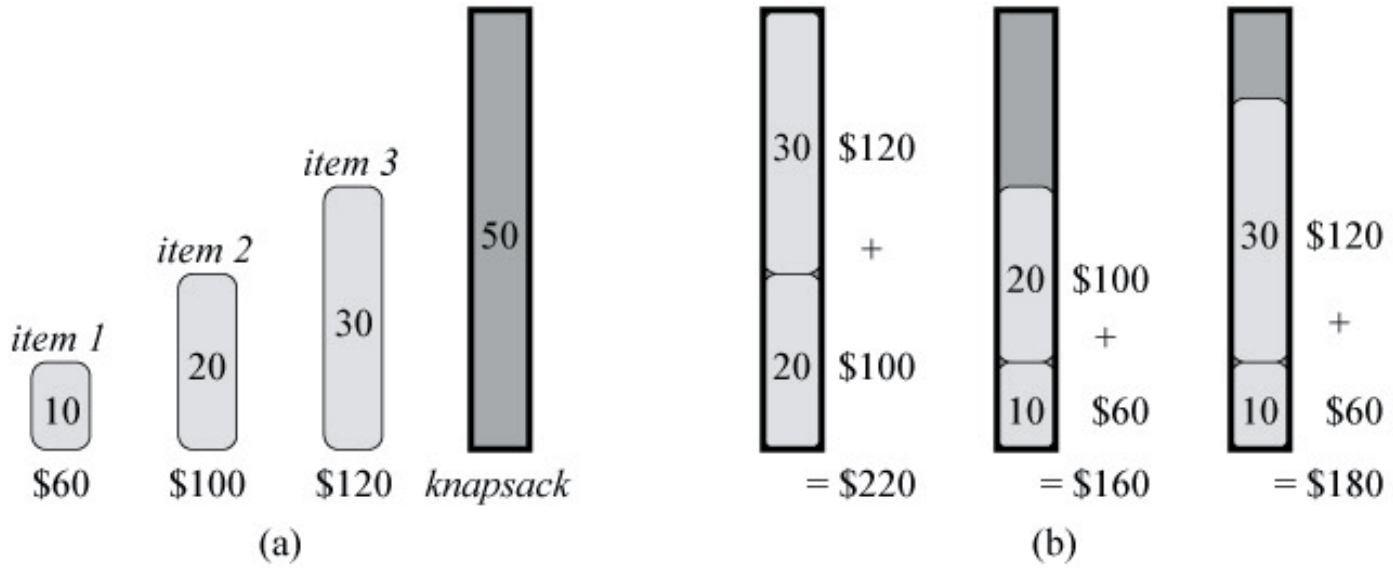
Lecture 12: 2D Dynamic Programming

The 0/1 Knapsack Problem



Input: A set of n items, where item i has weight w_i and value v_i, and a knapsack with capacity W.

Goal: Find $x_1, \ldots, x_n \in \{0,1\}$ such that $\sum_{i=1}^{n} x_i w_i \leq W$ and $\sum_{i=1}^{n} x_i v_i$ is maximized.

Recall: Greedy doesn't work

The Recurrence

Definition: Let $V[j]$ be the largest obtainable value for a knapsack with capacity j.

Recurrence:

$$V[j] = \max(0, v_1 + V[j - w_1], v_2 + V[j - w_2], \dots, v_n + V[j - w_n])$$
$$V[j] = 0, j \leq 0$$

This is wrong, since it may pick the same item more than once!

New definition: Let $V[i, j]$ be the largest obtained value for a knapsack with capacity j, only choosing from the first i items.

Recurrence:

$$V[i, j] = \max(V[i - 1, j], v_i + V[i - 1, j - w_i])$$
$$V[i, j] = 0, i = 0 \; or \; j = 0$$

The Algorithm

```
let V[0..n, 0..W] be a new array of all 0
for i ← 1 to n do
    for j ← 1 to W do
        if w[i] ≤ j and v[i] + V[i − 1, j − w[i]] > V[i − 1, j] then
            V[i, j] ← v[i] + V[i − 1, j − w[i]]
        else
            V[i, j] ← V[i − 1, j]
return V[n, W]
```

Running time: $\Theta(nW)$

Space: $\Theta(nW)$, but can be improved to $\Theta(n + W)$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

$V[i,j]$	0	1	2	3	4	5	6	7	8	9	10
$i = 0$	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

Reconstructing the Solution

Idea: Remember the optimal decision for each subproblem.

```
let V[0..n, 0..W] and keep[0..n, 0..W] be a new array of all 0
for i ← 1 to n do
    for j ← 1 to W do
        if w[i] ≤ j and v[i] + V[i − 1, j − w[i]] > V[i − 1, j] then
            V[i, j] ← v[i] + V[i − 1, j − w[i]]
            keep[i, j] ← 1
        else
            V[i, j] ← V[i − 1, j]
            keep[i, j] ← 0
K ← W
for i ← n downto 1 do
    if keep[i, K] = 1 then
        print i
        K ← K − w[i]
```

Running time: $\Theta(nW)$

Space: $\Theta(nW)$, cannot be improved to $\Theta(n + W)$ due to the $keep$ array.

Longest Common Subsequence

Problem: Given two sequences $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_n)$, we say that Z is a common subsequence of X and Y if Z has a strictly increasing sequence of indices i and j of both X and Y such that we have $x_{i_p} = y_{j_p} = z_p$ for all $p = 1, 2, \ldots, k$. The goal is to find the longest common subsequence of X and Y.

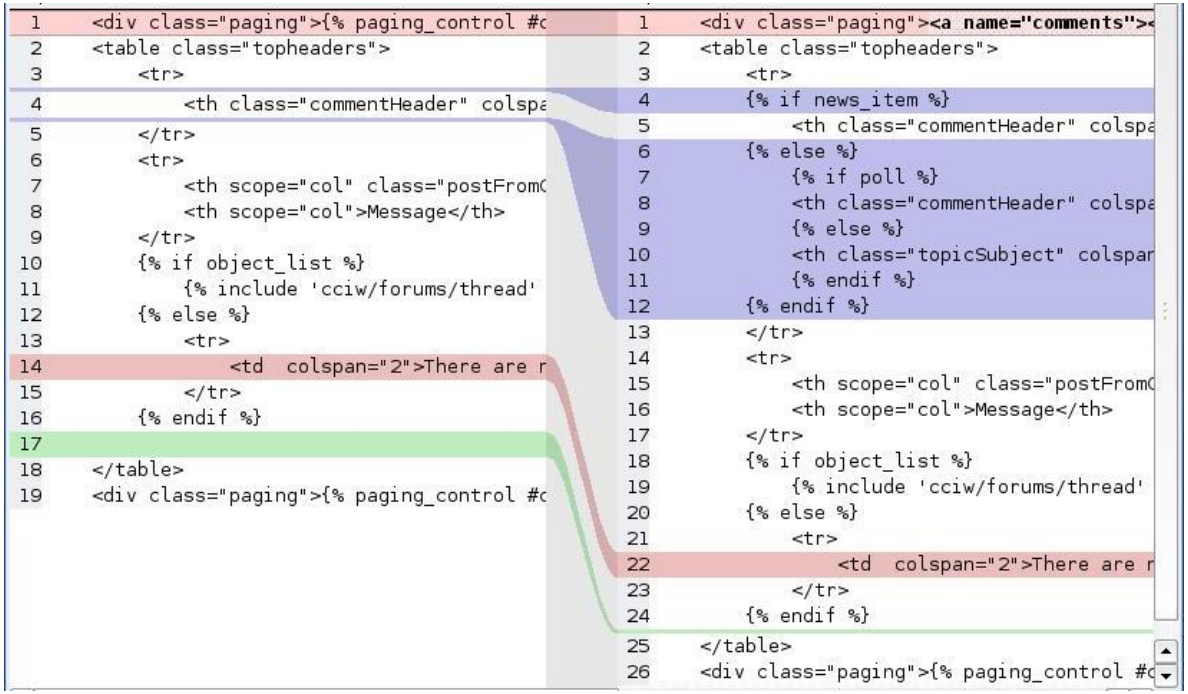
Ex:

X: **A B A C B D A B**

Y: **B D C A B A**

Z: **B C B A**

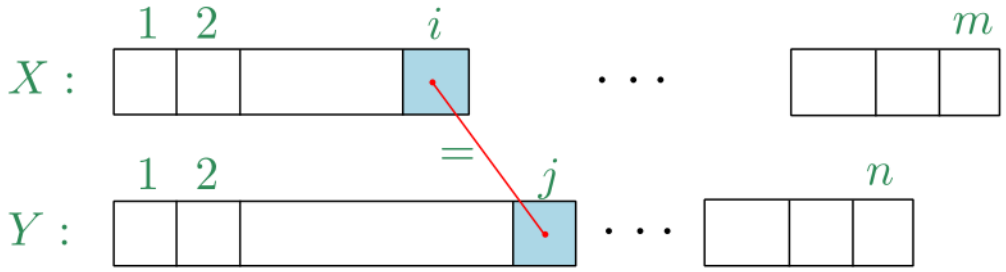
Application: diff



The Recurrence

Def: Let $c[i,j]$ to be the length of the longest common subsequence of $X[1..i]$ and $Y[1..j]$.

Observations: The problem is equivalent to finding the maximum matching between X and Y such that matched pairs don't cross.



The recurrence:

- Case 1: If $x_i = y_j$, then we match x_i and y_j. By doing so, we will not miss the optimal solution. (If OPT doesn't match them, we can change it so that they are matched.)
- Case 2: If $x_i \neq y_j$, then either x_i or y_j is not matched. So the problem reduces to either $c[i-1,j]$ or $c[i,j-1]$.

The Recurrence and Algorithm

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

```
let c[0..m, 0..n] and b[0..m, 0..n] be new arrays of all 0
for i ← 1 to m
    for j ← 1 to n
        if x_i = y_j then
            c[i,j] ← c[i − 1, j − 1] + 1
            b[i,j] ← " ↖ "
        else if c[i − 1, j] ≥ c[i, j − 1] then
            c[i,j] ← c[i − 1, j]
            b[i,j] ← " ↑ "
        else
            c[i,j] ← c[i, j − 1]
            b[i,j] ← " ← "
Print-LCS(b, m, n)
```

Running time: $\Theta(mn)$

Space: $\Theta(mn)$, can be improved to $\Theta(m+n)$ if we only need to return the optimal length.

Reconstruct the Optimal Solution

```
Print-LCS(b, i, j):
    if i = 0 or j = 0 then return
    if b[i,j] = "↖" then
        Print-LCS(b, i − 1, j − 1)
        print x_i
    else if b[i,j] = "↑"
        Print-LCS(b, i − 1, j)
    else Print-LCS(b, i, j − 1)
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Longest Common Substring

Problem: Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$, we wish to find their longest common substring Z, that is, the largest k for which there are indices i and j with $x_i x_{i+1} \ldots x_{i+k-1} = y_j y_{j+1} \ldots y_{j+k-1}$.

Ex:

X : DEADBEEF

Y : EATBEEF

Z : BEEF //pick the longest contiguous substring

Note: Brute-force algorithm takes $O(n^4)$ time.

The Recurrence

Def: $d[i,j] =$ the length of the longest common substring of $X[1..i]$ and $Y[1..j]$. (Does this work?)

Def: $d[i,j] =$ the length of the longest common substring of $X[1..i]$ and $Y[1..j]$ that ends at x_i and y_j.

Q: Wait, are we changing the problem?

A: Yes, but it's OK. The optimal solution to the original is just $\max\{d[i,j]\}$

Recurrence:

- If $x_i = y_j$, then the LCS of $X[1..i]$ and $Y[1..j]$ is just the LCS of $X[1..i-1]$ and $Y[1..j-1]$, plus $x_i = y_j$
- If $x_i \neq y_j$, then there can't be a common substring ending at x_i and y_j!

$$d[i,j] = \begin{cases} d[i-1,j-1]+1 & \text{if } x_i = y_j \\ 0 & \text{if } x_i \neq y_j \end{cases}$$

The Algorithm

```
let d[0..m, 0..n] be a new array of all 0
l_m ← 0, p_m ← 0
for i ← 1 to m
    for j ← 1 to n
        if x_i = y_j then
            d[i,j] ← d[i − 1, j − 1] + 1
            if d[i,j] > l_m then
                l_m ← d[i,j]
                p_m ← i
for i ← p_m − l_m + 1 to p_m
    print x_i
```

Note: For this problem, reconstructing the optimal solution just needs the location of the LCS.

Running time: $\Theta(mn)$

Space: $\Theta(mn)$ but can be improved to $\Theta(m+n)$.