Dynamic Programming Longest Common Subsequence

CSCI 3100

But first …

https://visualgo.net/bn/sorting

Follow up on the "sorting challenge" from last class

Which algorithm should we use to sort an "almost in order" array

Narrowed down to two options:

- Selection sort
- Merge sort

Dynamic Programming

An algorithm design technique similar to divide and conquer but unlike divide&conquer, subproblems may overlap in this case.

Divide and conquer
- Partition the problem into subproblems (may overlap)
- Solve the subproblems recursively
- Combine the solutions to solve the original problem

Used for **optimization problems**
- Goal: **find an optimal solution** (minimum or maximum)
- There may be many solutions that lead to an optimal value

Dynamic Programming

Applicable when subproblems are **not** independent
- Subproblems share subsubproblems

e.g.: Combinations:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

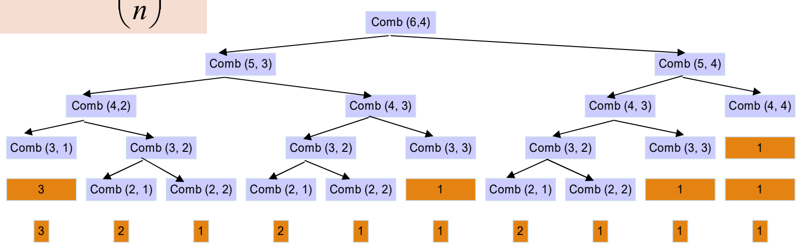
$$\binom{n}{1} = n \qquad \binom{n}{n} = 1$$

- Dynamic programming solves every subproblem and stores the answer in a table

Example: Combinations

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\binom{n}{1} = n \qquad \binom{n}{n} = 1$$



Dynamic Programming Algorithm

Characterize the structure of an optimal solution

Recursively define the value of an optimal solution

- An optimal solution to a problem contains within it an optimal solution to subproblems.
- Typically, the recursion tree contains many overlapping subproblems

Compute the value of an optimal solution in a bottom-up fashion

- Optimal solution to the entire problem is build in a bottom-up manner from optimal solutions to subproblems

Construct an optimal solution from computed information

Longest Common Subsequence

Given two sequences

$X = \langle x_1, x_2, \ldots, x_m \rangle$

$Y = \langle y_1, y_2, \ldots, y_n \rangle$

find a **maximum length common subsequence** (LCS) of X and Y

e.g.: If $X = \langle A, B, C, B, D, A, B \rangle$

Subsequences of X:

A subset of elements in the sequence taken in order

$\langle A, B, D \rangle$, $\langle B, C, D, B \rangle$, $\langle B, C, D, A, B \rangle$ etc.

Example

$X = \langle A, B, C, B, D, A, B \rangle$ $X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$ $Y = \langle B, D, C, A, B, A \rangle$

⟨B, C, B, A⟩ and **⟨B, D, A, B⟩** are
longest common subsequences of X and Y (*length* = 4)

$\langle B, C, A \rangle$, however, is not a LCS of X and Y

Applications of LCS

Molecular biology
- DNA sequences represented as combinations of letters ACGT
- Find how similar two sequences are

File comparison:
- Linux 'diff' command to compare two files

Brute-Force Solution

For every subsequence of X, check whether it's a subsequence of Y
- There are 2^m subsequences of X to check

Each subsequence takes $\Theta(n)$ time to check
- scan Y for first letter, from there scan for second, and so on

Running time: $\Theta(n2^m)$

10

Making the choice

X = ⟨A, B, D, G, E⟩

Y = ⟨Z, B, D, E⟩

Choice: include one element into the common sequence (E) and solve the resulting subproblem

X = ⟨A, B, D, G⟩

Y = ⟨Z, B, D⟩

Choice: exclude an element from a string and solve the resulting subproblem

Notations

Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$

we define the **i-th prefix** of X, for i = 0, 1, 2, …, m

$$X_i = \langle x_1, x_2, \ldots, x_i \rangle$$

c[i, j] = the **length** of a LCS of the sequences

$X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$

A Recursive Solution

Case 1: $x_i = y_j$

e.g.: $X_i = \langle A, B, D, G, E \rangle$

$Y_j = \langle Z, B, D, E \rangle$

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

- Append $x_i = y_j$ to the LCS of X_{i-1} and Y_{j-1}
- Must find a LCS of X_{i-1} and Y_{j-1}

A Recursive Solution

Case 2: $x_i \neq y_j$

e.g.: $X_i = \langle A, B, D, G \rangle$

$Y_j = \langle Z, B, D \rangle$

- Must solve two problems
  - find a LCS of X_{i-1} and Y_j: $X_{i-1} = \langle A, B, D \rangle$ and $Y_j = \langle Z, B, D \rangle$
  - find a LCS of X_i and Y_{j-1}: $X_i = \langle A, B, D, G \rangle$ and $Y_{j-1} = \langle Z, B \rangle$

$c[i, j] = \max \{ c[i - 1, j], c[i, j-1] \}$

Optimal solution to a problem includes optimal solutions to subproblems

Overlapping Subproblems

To find a LCS of (X_m and Y_n)

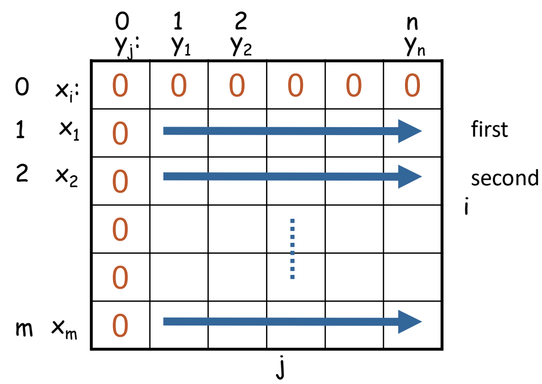
- we may need to find the LCS between $\mathbf{X_m}$ and $\mathbf{Y_{n-1}}$ and that of $\mathbf{X_{m-1}}$ and $\mathbf{Y_n}$
- Both of the above subproblems has the subproblem of finding the LCS of **($\mathbf{X_{m-1}}$ and $\mathbf{Y_{n-1}}$)**

Subproblems share subsubproblems

15

Computing the Length of the LCS

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



Additional Information

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

b & c:

		0	1	2	3		n
		y_j:	A	C	D		F
0	x_i	0	0	0	0	0	0
1	A	0					
2	B	0			c[i-1,j]		
3	C	0		↖ c[i,j-1]	↑		
		0					
m	D	0					

i

j

A matrix b[i, j]:

- For a subproblem [i, j] it tells us what choice was made to obtain the optimal value
- If $x_i = y_j$
  b[i, j] = "↖"
- Else, if c[i - 1, j] ≥ c[i, j-1]
  b[i, j] = " ↑ "
  else
  b[i, j] = " ← "

Example

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

X = ⟨A, B, C, B, D, A, B⟩

Y = ⟨B, D, C, A, B, A⟩

If $x_i = y_j$
  b[i, j] = "↖"
else if c[i - 1, j] ≥ c[i, j-1]
  b[i, j] = " ↑ "
else
  b[i, j] = " ← "

		0	1	2	3	4	5	6
		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

Constructing a LCS

Start at b[m, n] and follow the arrows

When we encounter a "↖" in b[i, j] ⇒ $x_i = y_j$ is an element of the LCS

		0	1	2	3	4	5	6
		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

LCS-LENGTH(X, Y, m, n)

```
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 to n
    do c[0, j] ← 0
for i ← 1 to m
    do for j ← 1 to n
        do 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] ← "←"
return c and b
```

If one of the sequences is empty, the length of the LCS is zero

Case 1: $x_i = y_j$

Case 2: $x_i \neq y_j$

Running time:

PRINT-LCS(b, X, i, j)

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

Initial call: PRINT-LCS(b, X, length[X], length[Y])

Running time:

Improving the Code

What can we say about how each entry c[i, j] is computed?

- It depends only on c[i -1, j - 1], c[i - 1, j], and c[i, j - 1]
- Eliminate table b and compute in $O(1)$ which of the three values was used to compute c[i, j]
- We save $\Theta(mn)$ space from table b
- However, we do not asymptotically decrease the auxiliary space requirements: still need table c

If we only need the length of the LCS

- LCS-LENGTH works only on two rows of c at a time
  - The row being computed and the previous row
- We can reduce the asymptotic space requirements by storing only these two rows