# Longest Common Subsequence

# Subsequences

- A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.
- In LCS ,we have to find Longest Common Subsequence that is in the same relative order.
- String of length n has 2^n different possible subsequences.
- E.g.—
- Subsequences of "ABCDEFG".
- "ABC","ABG","BDF","AEG",'ACEFG",.......

# Common Subsequences

Suppose that X and Y are two sequences over a set S.

X: ABCBDAB

Y: BDCABA

Z: BCBA

We say that Z is a common subsequence of X and Y if and only if

- Z is a subsequence of X
- Z is a subsequence of Y

# The Longest Common Subsequence Problem

Given two sequences X and Y over a set S, the longest common subsequence problem asks to find a common subsequence of X and Y that is of maximal length.

Z= (B,C,A) Length 3

Z= (B,C,A,B) Length 4

Z= (B,D,A,B) Length 4

Longest

# LCS Notation

Let X and Y be sequences.

We denote by LCS(X, Y) the set of longest common subsequences of X and Y.

LCS(X,Y)

Functional notation,

but not a function

# A Poor Approach to the LCS Problem

- A Brute-force solution:
  - Enumerate all subsequences of X
  - Test which ones are also subsequences of Y
  - Pick the longest one.
- Analysis:
  - If X is of length n, then it has $2^n$ subsequences
  - This is an exponential-time algorithm!

# Dynamic Programming

Let us try to develop a dynamic programming solution to the LCS problem.

# Optimal Substructure

Let $X = ( x_1, x_2, \ldots, x_m)$

and $Y = ( y_1, y_2, \ldots, y_n)$ be two sequences.

Let $Z = ( z_1, z_2, \ldots, z_k)$ is any LCS of X and Y.

a) If $x_m = y_n$ then certainly $x_m = y_n = z_k$

and $Z_{k-1}$ is in $LCS(X_{m-1}, Y_{n-1})$

# Optimal Substructure (2)

Let $X = (x_1, x_2, \ldots, x_m)$

and $Y = ( y_1, y_2, \ldots, y_n)$ be two sequences.

Let $Z = ( z_1, z_2, \ldots, z_k)$ is any LCS of X and Y.

b) If $x_m$ =! $y_n$ then $x_m$ =! $z_k$ implies that Z is in LCS($X_{m-1}$ , Y)

c) If $x_m$ =! $y_n$ then $y_n$ =! $z_k$ implies that Z is in LCS(X, $Y_{n-1}$)

# Recursive Solution

Let X and Y be sequences.

Let c[i,j] be the length of an element in LCS($X_i$, $Y_j$).

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

# Dynamic Programming Solution

- Define L[i,j] to be the length of the longest common subsequence of X[0..i] and Y[0..j].
- L[i,j-1] = 0 and L[i-1,j]=0, to indicate that the null part of X or Y has no match with the other.
- Then we can define L[i,j] in the general case as follows:
  1. If xi=yj, then L[i,j] = L[i-1,j-1] + 1 (we can add this match)
  2. If xi≠yj, then L[i,j] = max{L[i-1,j], L[i,j-1]} (we have no match here)

X:ABCB

Y:BDCA LCS:BC

# Dynamic Programming Solution (2)

How can we get an actual longest common subsequence?

Store in addition to the array c an array b pointing to the optimal subproblem chosen when computing c[i,j].

# Example

| T | $y_j$ | B | D | C | A |
|---|---|---|---|---|---|
| $X_i$ | 0 | 0 | 0 | 0 | 0 |
| A | 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↖ 1 |
| B | 0 | ↖ 1 | ← 1 | ← 1 | ↑ 1 |
| C | 0 | ↑ 1 | ↑ 1 | ↖ 2 | ← 2 |
| B | 0 | ↖ 1 | ↑ 1 | ↑ 2 | ↑ 2 |

Start at b[m,n]. Follow the arrows. Each diagonal array gives one element of the LCS.

# ALGORITHM LCS(X,Y)

```
m ← length[X]
n ← length[Y]
for i ← 1 to m do
      c[i,0] ← 0
for j ← 1 to n do
      c[0,j] ← 0
```

# LCS(X,Y)

```
for i ← 1 to m do
    for j ← 1 to n do
        if xi = yj
            c[i, j] ← c[i-1, j-1]+1
            b[i, j] ← "D"
        else
            if c[i-1, j] ≥ c[i, j-1]
                c[i, j] ← c[i-1, j]
                b[i, j] ← "U"
            else
                c[i, j] ← c[i, j-1]
                b[i, j] ← "L"
return c and b
```

# CONSTRUCTING AN LCS

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

1. If i=0 or j=0
2. then return
3. If b[i, j]="D"
4. then PRINT- LCS(b,X,i-1,j-1)
5. print xi
6. Else if b[i, j]="U"
7. then PRINT-LCS(b,X,i-1,j)
8. Else PRINT-LCS(b,X,i,j-1)

# Analysis of LCS Algorithm

- We have two nested loops
  - The outer one iterates *n* times
  - The inner one iterates *m* times
  - A constant amount of work is done inside each iteration of the inner loop
  - Thus, the total running time is $O(nm)$
- Answer is contained in L[n,m] (and the subsequence can be recovered from the T table).

# usage

- Biological applications often need to compare the DNA of two (or more) different organisms.
- We can say that two DNA strands are similar if one is a substring of the other.