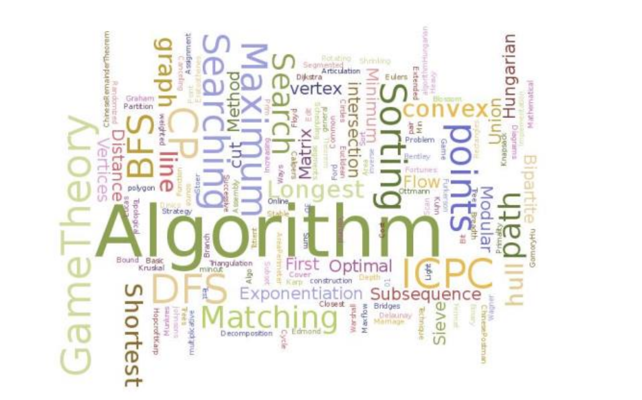
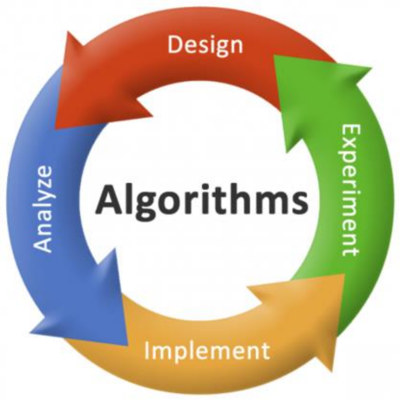
Dynamic Programming
Longest Common Subsequence

COP 3503
Fall 2021
Department of Computer Science
University of Central Florida
Dr. Steinberg





Applications

- Biologists often need to compare DNA of two or more different organisms
- DNA strand samples
  - ACCGGTCGATGCGVGGAAGCCGGCCGAA
  - GTCGTTCGGAATGCCGTTGCTCTGTAAA
- No Substring correlation !

Subsequences

- A subsequence is the given sequence with zero or more elements left out.
- Given a sequence X = $\langle x_1, x_2, \dots, x_m \rangle$ and $Z = \langle z_1, z_2, \dots, z_k \rangle$ we can a subsequence exists if there exists a strictly increasing sequence $\langle i_1, i_2, \dots, i_k \rangle$ of indices X such that for all $j = 1,2, \dots, k$ we have $x_i = z_j$
- For example $Z = \langle BCDB \rangle$ is a subsequence of X = $\langle ABCBDAB \rangle$ with corresponding index sequence $\langle 2,3,5,7 \rangle$

Common Subsequences

- Given 2 sequences X and Y, we say that a sequence Z is a **common subsequence** of X and Y if Z is a subsequence of both X and Y.
- Example
- X $= \langle A, B, C, B, D, A, B \rangle$ and Y $= \langle B, D, C, A, B, A \rangle$ the sequence $\langle B, C, A \rangle$ is a common subsequence.
- However it is not the longest common subsequence
- $\langle B, C, B, A \rangle$ is also another common subsequence with a length of 4. This is the longest common subsequence.

The Longest Common Subsequence Problem

- We are given two sequences X $= \langle x_1, x_2, \dots, x_m \rangle$ and Y $= \langle y_1, y_2, \dots, y_n \rangle$ and wish to find a maximum length common subsequences of X and Y.
- The objective is to find the MAXIMUM length common subsequence of X and Y.
- We will use Dynamic Programming!!

Computing the Length of LCS

Algorithm 1 LCS-Length (X, Y)

```
m = X.length
n = Y.length
for i = 0 to m do
    c[i,0] = 0
end for
for j = 1 to n do
    c[0,j] = 0
end for
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] =←
        end if
    end for
end for
return c, b
```

RT: O(mn)

Constructing the LCS

Algorithm 1 Print-LCS (b,X,i,j)

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

RT: O(m+n)

Example