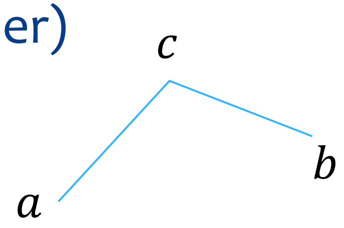
Dynamic Programming

High-level Idea: Break a complex problem into smaller (easier) subproblems subject to:



1. Principle of optimality (optimal substructure) –
   a substructure of an optimal structure is itself optimal.

   Example: A subpath of any shortest path is itself a shortest path.

2. Overlapping sub-problems: "many" smaller subproblems are actually the "same" problem.

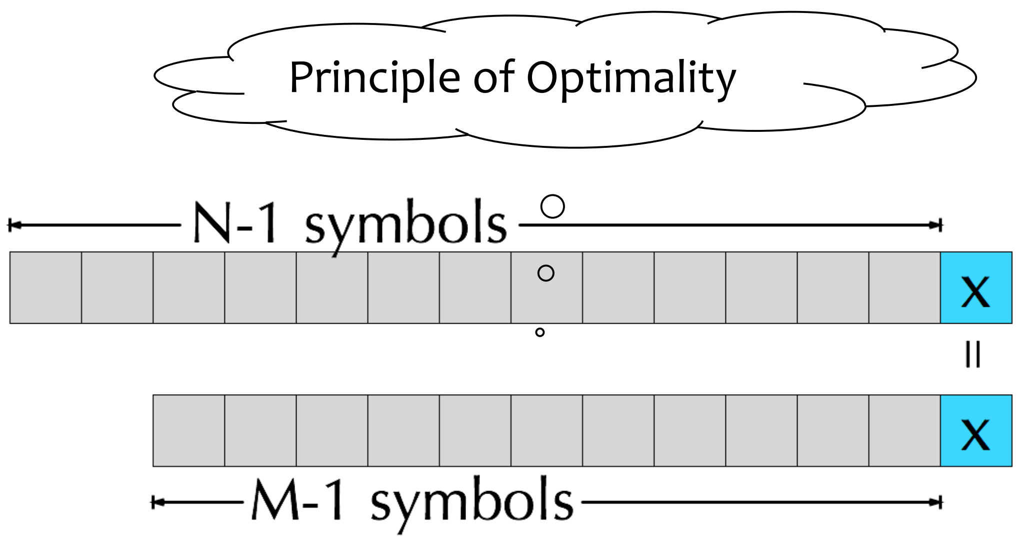
   Example: When computing the Fibonacci sequence using the rule:
   $F_n = F_{n-1} + F_{n-2}$, "many" recursive calls will be repeated.

Longest Common Subsequence

* **Definition:** A ***subsequence*** of a string s is a subset of the characters of s with respect to their original order.
  * **Example:** for s = "Fibonacci sequence"
    * "Fun"
    * "seen"
    * "cse"
    * ...
* Given strings $X[1..n]$ and $Y[1..m]$
* **Goal:** Find the length of a ***longest common subsequence*** of X and Y.
  * Largest string obtainable from X *and* Y by deleting chars
* **Example:** "Gole" is an LCS of "Google" and "Go Blue".

Longest Common Subsequence

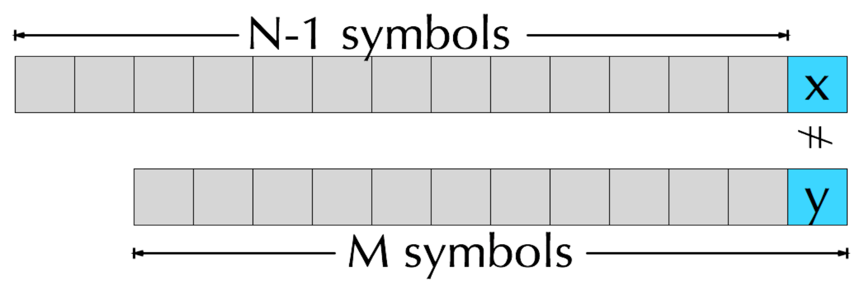
- **Idea:** Let X and Y be two strings of length n and m, respectively.
- $LCS(X[1..n], Y[1..m])$ = Length of LCS
- If the last characters are equal: $(X[n] = Y[m])$:
- $LCS(X[1..n], Y[1..m]) = LCS(X[1..n-1], Y[1..m-1]) + 1$



Longest Common Subsequence

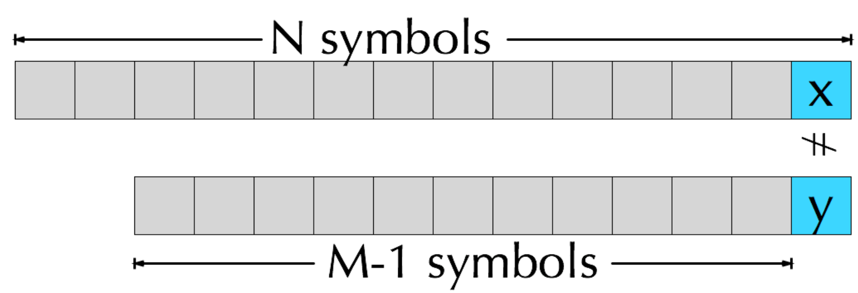
* **Idea:** Let X and Y be two strings of length n and m, respectively.
* If the last characters are **not** equal: $(X[n] \neq Y[m])$:
* $LCS(X[1..n], Y[1..m]) =$ Maximum of

$LCS(X[1..n-1], Y[1..m])$



and

$LCS(X[1..n], Y[1..m-1])$



Recurrence for LCS

- Let $LCS(i, j)$ denote the length of a longest common subsequence of $X[1..i]$ and $Y[1..j]$.
- Then:

$$LCS(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ 1 + LCS(i-1, j-1) & X[i] = Y[j] \\ \max\begin{Bmatrix} LCS(i-1, j), \\ LCS(i, j-1) \end{Bmatrix} & X[i] \neq Y[j] \end{cases}$$

Knapsack

- **Input:** $(v_1, s_1), \ldots, (v_n, s_n), B$
  - (v_i, s_i): value-size pair of item i.
  - B: size of bag
  - $B, s_1, \ldots, s_n$ are positive integers
- **Output:** $I \subseteq [n]$ s.t.
  - $\sum_{i \in I} s_i \leq B$ and
  - $\sum_{i \in I} v_i$ is maximized.
- Let $T[i, j] \coloneqq$ maximum value with back size j when we consider items $1, \ldots, i$.
  - $T[i, j] = \max(T[i-1, j], T[i-1, j-s_i] + v_i)$
    - (Second expression considered only if $j \geq s_i$)
  - Runtime: $O(nB)$