Problema da maior subsequência em comum:

Esse problema consiste em encontrar, dadas duas strings, a maior subsequência de caracteres em comum entre as duas. Note que podem existir diversas quantidades de subsequências entre duas strings, porém queremos a **ótima**, que neste caso se caracteriza por ser a mais longa.

Sem o uso de programação dinâmica, só é possível solucionar o problema através de tentativa e erro (força bruta), abordagem esta que possui um custo exorbitante. Veremos que, usando PD, conseguimos obter uma solução ótima com custo $O(n^2)$.

Consideremos as seguintes strings:
A = “ABCBDAB”
B = “BDCABA”

- **Etapa 1:** Caracterização da solução: A solução do problema para A[0..n] e B[0…m] envolve a solução de subproblemas como A[0…n-1] e B[0…m-1], A[0…n-2] e B[0…m-2] e assim por diante.
    -
- **Etapa 2:** Definir recursivamente o valor da solução: observando a etapa acima, é facil perceber a sobreposição de subproblemas, uma vez que ambos A[0…n-2] e B[0…m-1] e A[0…n-1] e B[0…m-2] levam ao subproblema A[0…n-2] e B[0…m-2].
    - ■
- **Etapa 3:** Calcular o valor da solução em abordagem *top-down* ou *bottom-up:* Vamos utilizar para esse problema uma abordagem bottom-up, de modo que começaremos resolvendo subproblemas triviais (A[0..0] e B[0…0]) e subiremos compondo a resposta final.

```
LCS-LENGTH (X, Y)
1   m = X.comprimento
2   n = Y.comprimento
3   sejam b[1..m, 1..n] e c[0..m, 0..n] tabelas novas
4   for i = 1 to m
5       c[i,0] = 0
6   for j = 0 to n
7       c[0, j] = 0
8   for i = 1 to m
9       for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j- 1]
17              b[i, j] = "←"
18  return c, b
```

Porém, a solução acima apenas indica o tamanho da subsequência mais longa, e não ela em si. Para isso utilizaremos um outro método em conjunto:

```
PRINT-LCS(b, X, i, j)
1  if i == 0 ou j == 0
2     return
3  if b[i, j] == "↖"
4     PRINT-LCS(b, X, i − 1, j − 1)
5     print x_i
6  elseif b[i, j] == "↑"
7     PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

Executando os algoritmos acima para a instância de exemplo, temos a seguinte tabela:

		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

Portanto, nossa solução ótima é “BCBA” e possui tamanho 4.

Sugestão de exercício: calcular a complexidade dos dois algoritmos citados.