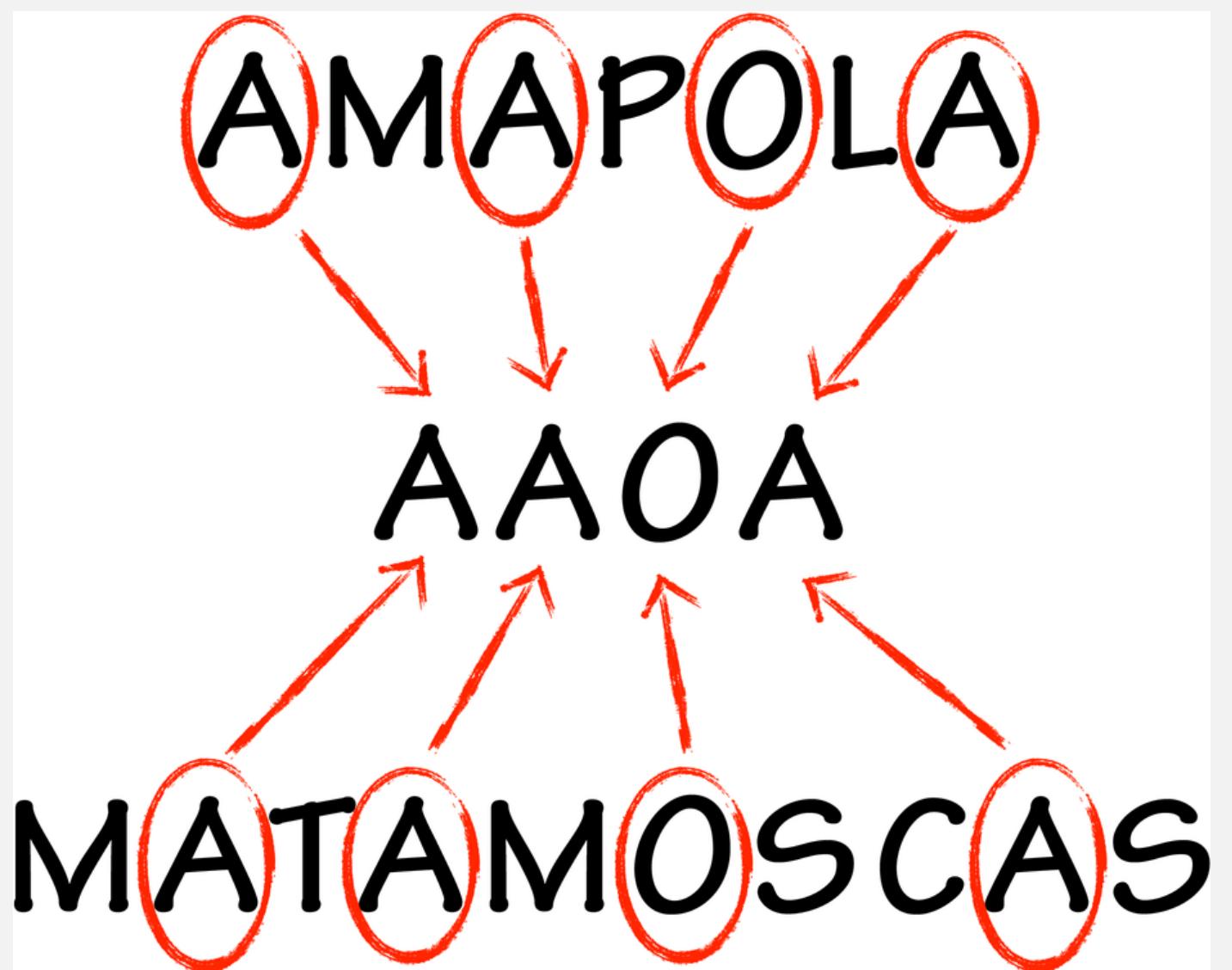


TECNICAS DE PROGRAMACION LINEAL

Valeria Larrea Guerrero

LA SUBSECUENCIA COMÚN MÁS LARGA.



Es un algoritmo que sirve para hallar la subsecuencia común más larga entre dos cadenas de caracteres.

El término también se aplica cuando quitamos todos los elementos (es decir la secuencia vacía es siempre subsecuencia de cualquier secuencia) o cuando no quitamos ninguno (lo que significa que cualquier secuencia es siempre subsecuencia de sí misma).

subsecuencia.py

distanzia.py

mochila.py

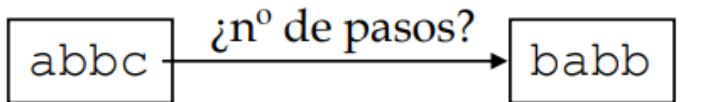
Extension: LiveCode for python

subsecuencia.py > ...

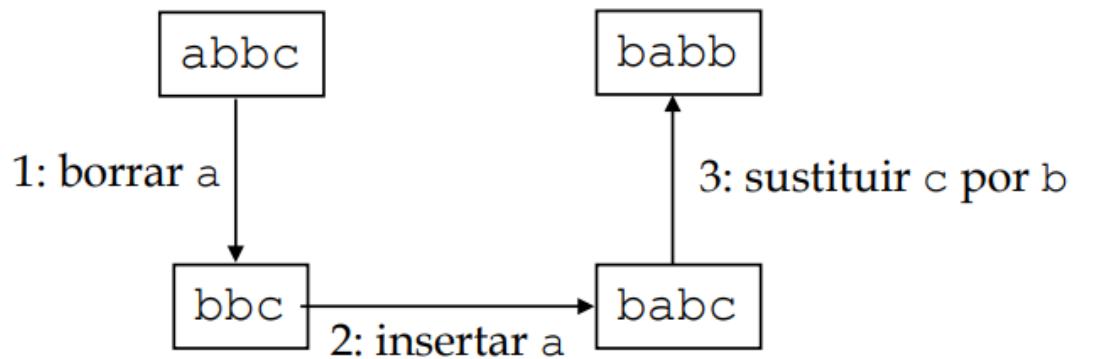
```
1  def subsecuencia(str1, str2):
2      m = len(str1)
3      n = len(str2)
4
5      dp = [[0] * (n + 1) for _ in range(m + 1)]
6
7      for i in range(1, m + 1):
8          for j in range(1, n + 1):
9              if str1[i - 1] == str2[j - 1]:
10                  dp[i][j] = dp[i - 1][j - 1] + 1
11              else:
12                  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
13
14      return dp[m][n]
15
16  cadena1 = input("Ingrese la primera cadena: ")
17  cadena2 = input("Ingrese la segunda cadena: ")
18  print("La longitud de la subsecuencia común más larga es:", subsecuencia(cadena1, cadena2))
19
```

DISTANCIA DE EDICIÓN

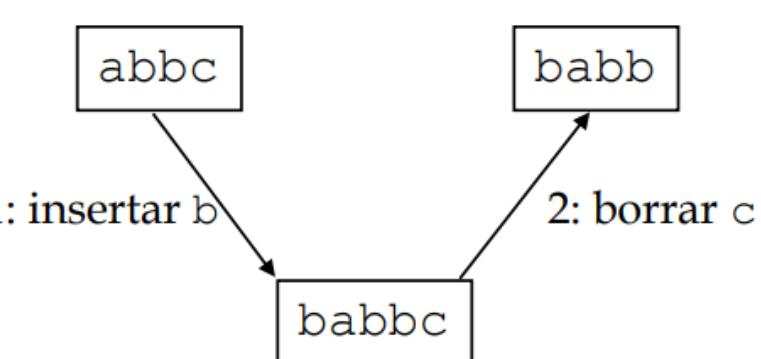
– Ejemplo:



Una solución:



Otra solución (mejor):



Es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Las operaciones de edición que se pueden hacer son:

- insertar un carácter (por ejemplo, de "abc" a "abca")
- eliminar un carácter (por ejemplo, de "abc" a "ac")
- sustituir un carácter (por ejemplo, de "abc" a "adc")

subsecuencia.py

distanzia.py X

mochila.py

Extension: LiveCode for python

distanzia.py > ...

```
1  def distancia(str1, str2):
2      m = len(str1)
3      n = len(str2)
4
5      dp = [[0] * (n + 1) for _ in range(m + 1)]
6
7      for i in range(m + 1):
8          for j in range(n + 1):
9              if i == 0:
10                  dp[i][j] = j
11              elif j == 0:
12                  dp[i][j] = i
13              elif str1[i - 1] == str2[j - 1]:
14                  dp[i][j] = dp[i - 1][j - 1]
15              else:
16                  dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])
17
18      return dp[m][n]
19
20  cadena1 = input("Ingrese la primera cadena: ")
21  cadena2 = input("Ingrese la segunda cadena: ")
22  print("La distancia de edición es:", distancia(cadena1, cadena2))
23
```

n : Número de objetos distintos

x_i : Elemento i

w_i : Peso/volumen del elemento i

b_j : Beneficio/Valor del elemento i

c : Capacidad total de la mochila

y sean c , w_i y b_i enteros positivos

Podemos plantear el modelo como

Maximizar $\sum_{i=1}^n b_i x_i$

sujeto a $\sum_{i=1}^n w_i x_i \leq c$

EL PROBLEMA DE LA MOCHILA.

El problema de la mochila es un ejercicio típico de la rama de matemáticas que se llama «Investigación Operativa».

Consiste en un excursionista que debe preparar su mochila, la cual tiene una capacidad limitada, esto hace que no pueda incluir todas las cosas que este en un primer momento querría llevar a la excursión.

```
def mochila(weights, values, capacity):
    n = len(weights)

    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]

n = int(input("Ingrese el número de elementos: "))
pesos = []
valores = []
for i in range(n):
    peso = int(input(f"Ingrese el peso del elemento {i + 1}: "))
    valor = int(input(f"Ingrese el valor del elemento {i + 1}: "))
    pesos.append(peso)
    valores.append(valor)
capacidad = int(input("Ingrese la capacidad de la mochila: "))
print("El valor máximo que se puede obtener es:", mochila(pesos, valores, capacidad))
```