

## Taller 2 - Programación Dinámica

David Gutierrez Alarcon,  
Julian Andres Carrillo Chiquisa,  
David Alejandro Castillo Chiquiza

26 de septiembre de 2021

### Resumen

En este documento se presenta el análisis de los dos problemas planteados y sus soluciones mediante el uso de el método de Programación Dinámica

## Análisis y Diseño del Problema

### Análisis

Para el desarrollo del ejercicio, es necesario que se dispongan de dos cadenas  $X$  y  $Y$  de  $m$  y  $n$  caracteres respectivamente, con el fin de determinar si el algoritmo es capaz de barajar las dos secuencias de elementos anteriores, de una forma determinada por una cadena  $Z$  dada previamente, esto implica que la cadena barajada debe estar conformada tomando todos los elementos de  $X$  y  $Y$ , con orden de  $Z$ , si embargo, no necesariamente deben ser contiguos. El problema se puede modelar de la siguiente manera:

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle$$

$$Z = \langle z_1, z_2, \dots, z_{m+n} \rangle = \langle z_i \in T \mid 1 \leq i \leq m+n \rangle$$

Donde  $n$  y  $m$  son la cantidad respectiva de los conjuntos  $X$  y  $Y$ , dando a entender que  $m+n$  son todos los elementos de  $Z$  con  $z_i$  como los elementos pertenecientes al conjunto  $T$ .

### Diseño

El Algoritmo debe validar si es posible generar una secuencia con los elementos de  $X$  y  $Y$  que correspondan con los elementos de  $Z$ .

## Redursivo Evidente

### Entradas

- Una Secuencia  $X = \langle x_1, x_2, \dots, x_m \rangle$
- una secuencia  $Y = \langle y_1, y_2, \dots, y_n \rangle$
- Una secuencia  $Z = \langle z_1, z_2, \dots, z_{m+n} \rangle = \langle z_i \in T \mid 1 < i \leq n + m \rangle$
- Una secuencia  $A$  vacia
- Una secuencia copia de  $Z$  de nombre  $Q$

### Salidas

- Un dato Booleano que determina si es posible la generación de la secuencia  $Z$

## Memoizado

### Entradas

- Una Secuencia  $X = \langle x_1, x_2, \dots, x_m \rangle$
- una secuencia  $Y = \langle y_1, y_2, \dots, y_n \rangle$
- Una secuencia  $Z = \langle z_1, z_2, \dots, z_{m+n} \rangle = \langle z_i \in T \mid 1 < i \leq n + m \rangle$
- Valor  $m$  que hace referencia al tamaño de la longitud de  $X$
- Valor  $n$  que hace referencia al tamaño de la longitud de  $Y$
- Matriz  $C$  vacia de tamaño  $m, n$

### Salidas

- Secuencia de Carcacteres correspondientes a  $Z$  en el caso dado donde si sea posible generar la secuencia de caracteres, en caso contrario, se presenta una secuencia con la mayor cantidad de caracteres posibles similares a  $Z$

Bottom-up

Entradas

Salidas

# Algoritmos

Evidente recursivo

Pseudocódigo

---

**Algorithm 1** EvidenteRecursivo

---

```
1: procedure EVIDENTERECURSIVO( $X, Y, Z, A, Q$ )
2:   if  $A == Q$  then
3:      $s \leftarrow \text{True}$ 
4:     return  $A$ 
5:   end if
6:   if  $s == \text{False}$  then
7:     if  $(\text{longitud}(X) == 0) \wedge (\text{longitud}(Y) == 0)$  then
8:       return  $A$ 
9:     end if
10:    if  $(\text{longitud}(X) > 0) \vee (\text{longitud}(Y) > 0)$  then
11:      if  $\text{longitud}(X) > 0$  then
12:        if  $X[0] == Z[0]$  then
13:           $\text{EvidenteRecursivo}(X[1:], Y, Z[1:], A + X[0], Q)$ 
14:        end if
15:      end if
16:      if  $\text{longitud}(Y) > 0$  then
17:        if  $Y[0] == Z[0]$  then
18:           $\text{EvidenteRecursivo}(X, Y[1:], Z[1:], Z + Y[0], Q)$ 
19:        end if
20:      end if
21:    end if
22:  end if
23:  return  $A$ 
24: end procedure
```

---

## Complejidad

Para este algoritmo, se tiene que revisar todas las combinaciones de las secuencias  $X$  y  $Y$ , por lo tanto la complejidad es  $O(2^{mn})$

## **Invariante**

Todos los elementos de  $X$  y  $Y$  se agregan a la lista  $A$ , cumpliendo con la propiedad de orden y pueden o no estar en un orden consecutivo

### **inicio**

- Lista  $A$  vacia

### **Avance**

- Si el primer item del arreglo  $X$  ó  $Y$  coincide con el primer elemento de  $Z$ , dicho elemnto se agrega a la lista de  $A$

### **Terminación**

- Termina cuando la lista de  $A$  es igual a la lista de  $Q$  y retorna verdadero
- Termina cuando no se encuentren caracteres en  $X$  y  $Y$  que coincidan con los carcateres de la lista  $Z$  y retorna Falso

## **Notas de Implementación**

La implementación del pseudocodigo fue realizada en el lenguaje Python y se puede encontrar en el archivo adjunto a este documento, El nombre del archivo es: recursivoInocente.ipynb.

# Memoizado

## Pseudocódigo

---

**Algorithm 2** Memorizado

---

```
1: procedure MEMOIZADO( $X, Y, Z, M, N, C$ )
2:   if  $C[M][N] \neq ""$  then
3:     return  $C[M][N]$ 
4:   end if
5:   if  $longitud(z) == 0$  then
6:      $sePudo = True$ 
7:     return  $C[M][N]$ 
8:   end if
9:   if  $longitud(X) > 0 \vee longitud(Y) > 0$  then
10:    if  $longitud(X) > 0$  then
11:      if  $X[0] == Z[0] \wedge sePudo == False$  then
12:         $C[M][N] = str(Memoizado(X[1:], Y, Z[1:], M - 1, N, C)) +$ 
13:         $X[0]$ 
14:      end if
15:    end if
16:    if  $longitud(Y) > 0$  then
17:      if  $Y[0] == Z[0] \vee sePudo == False$  then
18:         $C[M][N] = str(Memoizado(X, Y[1:], Z[1:], M, N - 1, C)) +$ 
19:         $Y[0]$ 
20:      end if
21:    end if
22:  end if
23:  return  $C[M][N]$ 
24: end procedure
```

---

## Complejidad

La complejidad se determina teniendo en cuenta que se genera una matriz de tamaño  $m, n$  y haciendo uso de la Memiozación podemos decir que la complejidad del algoritmo en  $O(n^2)$

## Invariante

### inicio

- Matriz  $C$  vacia

### Avance

- Si la matriz en la posicion  $m, n$  no está vacia, retorna el contenido en dicha posición.

- Si el primer item del arreglo  $X$  ó  $Y$  coincide con el primer elemento de  $Z$ , dicho elemnto se agrega a la matriz  $C$  en la posición  $m, n$ .

#### **Terminación**

- Termina cuando el arreglo de  $Z$  está vacío y retorna la matriz  $C$  en la posición  $m, n$ .
- Termina cuando no se encuentren caracteres en  $X$  y  $Y$  que coincidan con los caracteres de la lista  $Z$  y retorna la matriz  $C$  en la posición  $m, n$ .

#### **Notas de Implementación**

La implementación del pseudocódigo fue realizada en el lenguaje Python y se puede encontrar en el archivo adjunto a este documento, El nombre del archivo es: recursivoMemoizado.py

### **Bottom-up**

#### **Pseudocódigo**

#### **Complejidad**

#### **Invariante**

#### **inicio**

#### **Avance**

#### **Terminación**

#### **Notas de Implementación**

## **Comparación**

Para comparar los algoritmos, se generaron 3 casos distintos tomando como variable a evaluar, los tiempos de ejecución de cada algoritmo. Para realizar la comparación, no se tuvieron en cuenta las impresiones en consola con el fin de no estropear las medidas de tiempo de ejecución.

Resultados del Primer Algoritmo:

---

```
True
True
True
--- 0.01451253890991211 tiempo de finalizacion ---
tiempo caso 1: 0.0004820823669433594
tiempo caso 2: 0.011086225509643555
tiempo caso 3: 0.01446843147277832
```

Resultados del Segundo Algoritmo:

---

```
True
CCASARROA
True
ababababababababababababababab
True
aassddffassddffhsedrataasadsadsadsadsadsads
--- 0.003795146942138672 seconds ---
tiempo caso 1: 0.0013861656188964844
tiempo caso 2: 0.0021843910217285156
tiempo caso 3: 0.0037641525268554688
```

Resultados del Tercer Algoritmo: