


Estructura de Datos y Algoritmos

ITBA 2024-Q2

Levenshtein Distance

Es un algoritmo que calcula la **MINIMA** cantidad de operaciones necesarias para transformar un string en otro. Las operaciones válidas son: **insertar, borrar o sustituir un caracter.**

Aquellos strings que son iguales, deben tener distancia 0 porque no hace falta transformar uno en otro.

Ej: Levenshtein('big data', 'bigdata') = 

1 es la mínima, pero hay muchas formas de ir de str1 a str2.

'big data' => BBBBIII o sea 7

'big data' => BSSS o sea 4

'big data' => ---SSSSD o sea 5

Etc, etc, etc.

Levenshtein Distance

A diferencia de Soundex, que se adaptó para proponer una medida de similitud, a partir de su cálculo, **Levenshtein ES** un métrica de distancia. Por lo tanto, cumple con propiedades:

- a) **Simetría:** $\text{Levenshtein}(\text{str1}, \text{str2}) = \text{Levenshtein}(\text{str2}, \text{str1})$
- b) **Desigualdad:** $\text{Levenshtein}(\text{str1}, \text{str2}) + \text{Levenshtein}(\text{str2}, \text{str3}) \geq \text{Levenshtein}(\text{str1}, \text{str3})$

El problema es que una implementación ensayo&error sería tan ineficiente!
Impracticable.

Por eso, se lo suele implementar con la
técnica de Programación Dinámica

Levenshtein Distance

Programación Dinámica es una técnica que consiste en reusar valores previamente calculados para no tener que recalcularlos repetidamente. Sirve, si para cierto cálculo, pueden reusarse valores previos...

Así, los valores calculados deben almacenarse en una estructura de datos (vector, matriz, etc.) con el objetivo de "buscarlos" (lookup) y no calcularlos nuevamente cuando se los precise.

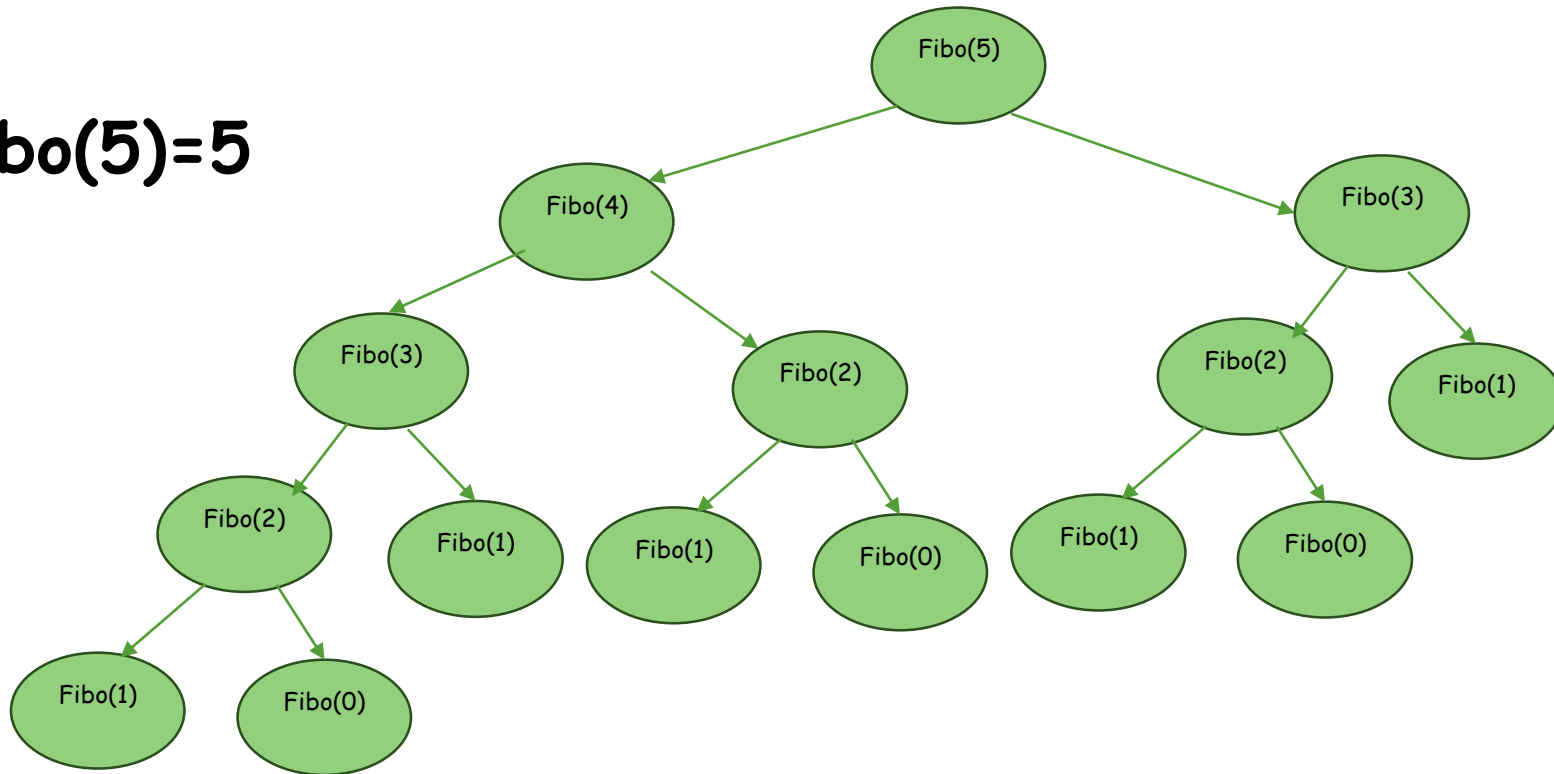
Desde el punto de vista de complejidad algorítmica computacional, una operación Lookup podría ser de costo bajísimo!

¿Conocen algún ejemplo de uso de programación dinámica?

Cálculo de camino más corto entre 2 nodos de un grafo, Fibonacci, etc, etc.

EJ: $\text{Fibo}(N) = N$ si $N=0$ or $N= 1$
 $\text{Fibo}(N-1) + \text{Fibo}(N-2)$ si $N > 1$

$\text{Fibo}(5)=5$



$$\text{EJ: Fibo}(N) = \begin{cases} N & \text{si } N=0 \text{ or } N= 1 \\ \text{Fibo}(N-1) + \text{Fibo}(N-2) & \text{si } N > 1 \end{cases}$$

$$\begin{aligned} \text{Fibo}(5) &= \text{Fibo}(4) + \text{Fibo}(3) \\ &= \text{Fibo}(3) + \text{Fibo}(2) + \text{Fibo}(2) + \text{Fibo}(1) \\ &= \text{Fibo}(2) + \text{Fibo}(1) + \text{Fibo}(1) + \text{Fibo}(0) + \text{Fibo}(1) + \text{Fibo}(0) + 1 \\ &= \text{Fibo}(1) + \text{Fibo}(0) + 1 + 1 + 0 + 1 + 0 + 1 \\ &= 1 + 0 + 1 + 1 + 0 + 1 + 0 + 1 = 5 \end{aligned}$$

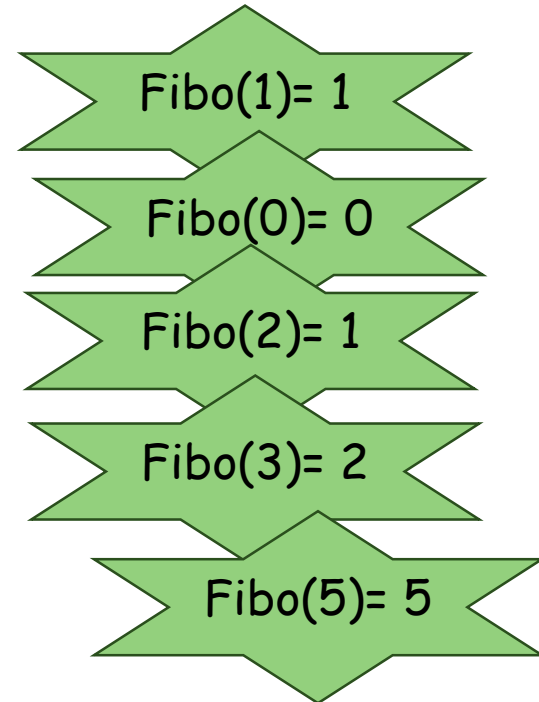
El problema es que este algoritmo recursivo tiene complejidad temporal $O(2^N)$.

La complejidad espacial es $O(N)$

EJ: $\text{Fibo}(N) = N$ si $N=0$ or $N= 1$
 $\text{Fibo}(N-1) + \text{Fibo}(N-2)$ si $N > 1$

Con programación dinámica, los valores calculados los aprovecharíamos.

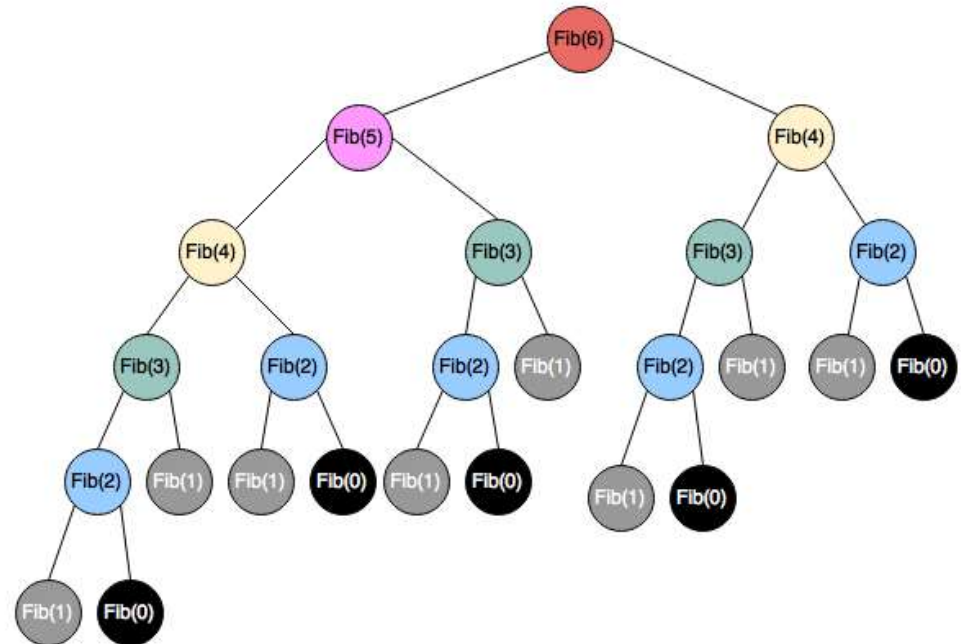
$$\begin{aligned}
 \text{Fibo}(5) &= \text{Fibo}(4) && + \text{Fibo}(3) \\
 &= \text{Fibo}(3) + && \text{Fibo}(2) + ? \\
 &= \text{Fibo}(2) + && \text{Fibo}(1) + ? && + ? \\
 &= \text{Fibo}(1) + \text{Fibo}(0) + 1 && + ? && + ? \\
 &= 1 + 0 + 1 && + ? && + ? \\
 &= 1 + 0 + 1 && + 1 && + ? \\
 &= 1 + 0 + 1 && + 1 && + 2 = 5
 \end{aligned}$$



Este algoritmo es $O(N)$.

Levenshtein Distance

$$\text{EJ: Fibo}(N) = \begin{cases} N & \text{si } N=0 \text{ or } N= 1 \\ \text{Fibo}(N-1) + \text{Fibo}(N-2) & \text{si } N > 1 \end{cases}$$

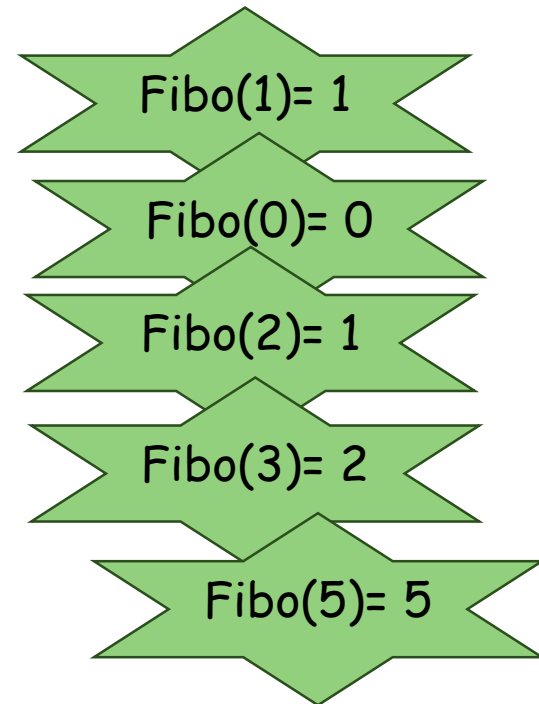


El problema es que este algoritmo recursivo es $O(2^N)$.

EJ: $\text{Fibo}(N) = N$ si $N=0$ or $N= 1$
 $\text{Fibo}(N-1) + \text{Fibo}(N-2)$ si $N > 1$

Con programación dinámica, los valores calculados los aprovecharíamos.

$$\begin{aligned}
 \text{Fibo}(5) &= \text{Fibo}(4) && + \text{Fibo}(3) \\
 &= \text{Fibo}(3) + && \text{Fibo}(2) + ? \\
 &= \text{Fibo}(2) + && \text{Fibo}(1) + ? && + ? \\
 &= \text{Fibo}(1) + \text{Fibo}(0) + 1 && + ? && + ? \\
 &= 1 + 0 + 1 && + ? && + ? \\
 &= 1 + 0 + 1 && + 1 && + ? \\
 &= 1 + 0 + 1 && + 1 && + 2 = 5
 \end{aligned}$$



Este algoritmo es $O(N)$.

Posible implementación (si invocan con $N \geq 50$ se nota la diferencia con cache o sin cache)

```
public class FastFibo {  
    private static HashMap<Integer, Long> cache= new HashMap<>();  
  
    public static long fibo(int N) {  
        if (cache.containsKey(N))  
            return cache.get(N);  
  
        long rta;  
  
        if (N <= 1)  
            rta= N;  
        else  
            rta= fibo(N-1) + fibo(N-2);  
  
        cache.put(N, rta);  
  
        return rta;  
    }  
}
```

String Matching – Levenshtein Distance

Programa 'big data',

	♥	B	I	G		D	A	T	A
♥									
B									
I									
G									
D									
A									
T									
A									

La celda representa
Levenshtein('BIG', 'BI')

Levenshtein Distance

Programación Dinámica para Levenshtein('big data', 'bigdata')

	♥	B	I	G		D	A	T	A
♥	0	1	2	3	4	5	6	7	8
B	1								
I	2								
G	3								
D	4								
A	5								
T	6								
A	7								

Entonces
Levenshtein("", 'B')= 1
Levenshtein("", 'BI')=2
Levenshtein("", 'BIG')=3
Levenshtein("", 'BIGD')=4
...

Entonces
Levenshtein("", "")= 0
Levenshtein('B', "")=1
Levenshtein('BI', "")=2
Levenshtein('BIG', "")=3
...

Levenshtein Distance

Programación Dinámica para Levenshtein('bia data', 'biadata')

	♥	B	I	G		D	A	T	A
♥	0	1	2	3	4	5	6	7	8
B	1								
I	2								
G	3								
D	4								
A	5								
T	6								
A	7								

La celda representa
Levenshtein('BIG', 'BI')

Levenshtein('BIG', 'BI')=
Min (Levenshtein('BI', 'B') + 'G'=='I'?0:1,
 Levenshtein('BIG', 'B') + 1
 Levenshtein('BI', 'BI') + 1,
)

Levenshtein Distance

Programación Dinámica para Levenshtein('big data', 'bigdata')

	♥	B	I	G		D	A	T	A
♥	0	1	2	3	4	5	6	7	8
B	1								
I	2								
G	3								
D	4								
A									
T									
A									

Levenshtein($\alpha w, \chi z$) =

Min (



)



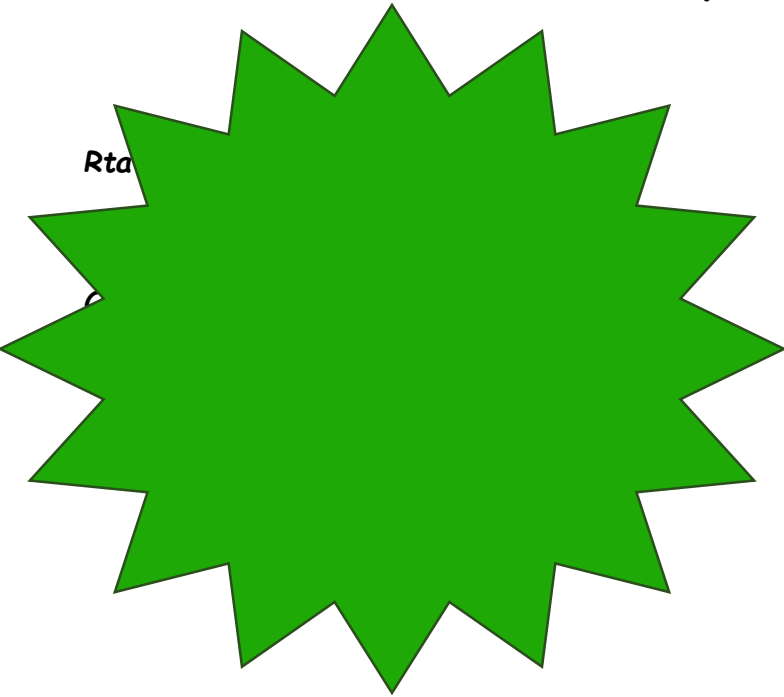
Levenshtein Distance

Calculamos la distancia de Levenshtein('big data', 'bigdata')

	♥	B	I	G		D	A	T	A	
♥	0	1	2	3	4	5	6	7	8	
B	1									
I	2									
G	3									
D	4									
A	5									
T	6									
A	7									

Levenshtein Distance

¿Cuál es la distancia Levenshtein ("exkusa", "ex-amigo") ?



Levenshtein Distance

Se puede normalizar para que el número obtenido esté entre 0 y 1. El valor 1 implica coincidencia.

$$\text{LevenshteinNormalized}(\text{str1}, \text{str2}) = 1 - \frac{\text{Levenshtein}(\text{str1}, \text{str2})}{\max(\text{str1.len}, \text{str2.len})}$$

Levenshtein Distance

Existen variantes:

Por ejemplo: Damerau-Levenshtein: las operaciones no son sólo borrado, inserción, y sustitución. También se agrega transposición.

Otras variantes no consideran que las operaciones valen todas igual. Alguna es más cara que otra y cambia la fórmula de distancia, entonces.