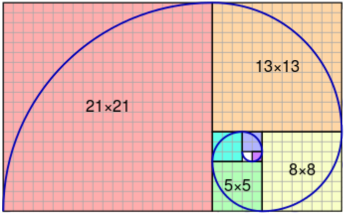
Técnicas Algorítmicas

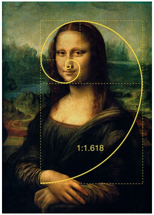
Prof. Agustín Gravano

MiM - UTDT - Segundo semestre de 2020

Clase 3: Programación Dinámica

Número de Fibonacci





$$Fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{si } n > 1 \end{cases}$$

Sucesión de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Número de Fibonacci

```
def fib(n):
  if n==0:
    res = 0
  elif n==1:
    res = 1
  else:
    res = fib(n-1) + fib(n-2)
  return res
```

Algoritmo muy simple y elegante, pero...

$T(0) = O(1)$

$T(1) = O(1)$

$T(n) = T(n-1) + T(n-2) + O(1) \quad (n > 1)$

Es sencillo ver que $T(n) \in O(2^n)$ 

Recursión

La recursión es útil para escribir código simple y claro, pero hay que tener mucho cuidado con la complejidad temporal.

La recursión no es mala o buena *per se*.

Sólo hay que tener cuidado.

A veces nos lleva a algoritmos ineficientes (ej: Fibonacci) y otras veces a algoritmos eficientes (ej: mergesort).

Ejercicio para resolver ahora

Sucesión de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Escribir una función iterativa `fib` que, dado un número entero $n \geq 0$, devuelva el n-ésimo número de Fibonacci, en tiempo lineal respecto de n.

Ejemplos:

- `fib(0)` $\longmapsto 0$
- `fib(1)` $\longmapsto 1$
- `fib(2)` $\longmapsto 1$
- `fib(3)` $\longmapsto 2$
- `fib(4)` $\longmapsto 3$
- ...

Programación Dinámica

Objetivo:

- Evitar la repetición de cómputos.

Idea:

- Ir almacenando en una estructura de datos (p.ej., una lista, una matriz) resultados parciales que puedan ser necesarios más adelante.

Distancia de edición entre 2 strings

(También conocida como Distancia de Levenshtein)

¿Cuántas operaciones son necesarias para transformar un string en otro?

Operaciones válidas: inserción y eliminación, siempre de a un caracter por vez.
(La formulación que incluye sustitución es equivalente; sólo un poco más compleja.)

Ejemplo: La distancia de edición entre `AGUA` y `GOTA` es 4.

0. `AGUA`
1. `GUA` (eliminación)
2. `GOUA` (inserción)
3. `GOA` (eliminación)
4. `GOTA` (inserción)

Ejemplo: ALGORITMO $\xrightarrow{?}$ PROGRAMACION

Sup. que sabemos que ALGORITM $\xrightarrow{14}$ PROGRAMACION:

$$\text{ALGORITMO} \xrightarrow{14} \text{PROGRAMACIONO} \xrightarrow{1} \text{PROGRAMACION}$$

Entonces, por este camino, que culmina en la eliminación de la O:

$$\text{ALGORITMO} \xrightarrow{15} \text{PROGRAMACION}$$

Si además sabemos que ALGORITMO $\xrightarrow{12}$ PROGRAMACIO:

$$\text{ALGORITMO} \xrightarrow{12} \text{PROGRAMACIO} \xrightarrow{1} \text{PROGRAMACION}$$

Entonces, por este camino, que culmina en la inserción de la N:

$$\text{ALGORITMO} \xrightarrow{13} \text{PROGRAMACION}$$

De estos dos caminos, nos quedamos con el mínimo: 13.

Observación:

$$\begin{gathered}\text{ALGORITM} \xrightarrow{14} \text{PROGRAMACION}\\ \text{ALGORITMO} \xrightarrow{12} \text{PROGRAMACIO}\end{gathered}$$

son problemas más pequeños que el problema original:

$$\text{ALGORITMO} \xrightarrow{?} \text{PROGRAMACION}$$

Esto podría servirnos para pensar un algoritmo recursivo para calcular la distancia de edición entre 2 strings.

Otro ejemplo: PROBLEMA $\xrightarrow{?}$ PROGRAMA

Si sabemos que PROBLEM $\xrightarrow{7}$ PROGRAMA, entonces:

$$\text{PROBLEMA} \xrightarrow{7} \text{PROGRAMAA} \xrightarrow{1} \text{PROGRAMA}$$

Si sabemos que PROBLEMA $\xrightarrow{7}$ PROGRAM, entonces:

$$\text{PROBLEMA} \xrightarrow{7} \text{PROGRAM} \xrightarrow{1} \text{PROGRAMA}$$

Estos dos caminos tienen 8 operaciones.

Pero este caso tiene una diferencia importante con el anterior: PROBLEMA y PROGRAMA terminan en la misma letra (A).

Entonces, si sabemos que PROBLEM $\xrightarrow{6}$ PROGRAM, entonces podríamos usar solo 6 operaciones en total:

$$\text{PROBLEMA} \xrightarrow{6} \text{PROGRAMA}$$

De estos tres caminos posibles, nos quedamos con el mínimo: 6.

Distancia de edición entre 2 strings

Pensemos un algoritmo recursivo.

Casos base: ('' representa al string vacío)

`ED(s, '')` = `len(s)` (todas eliminaciones)

`ED('', t)` = `len(t)` (todas inserciones)

Si un string es vacío, la cantidad necesaria de operaciones (todas inserciones, o bien todas eliminaciones) es la longitud del otro string.

```
def ED(s, t):
  if len(s)==0:
    res = len(t)
  elif len(t)==0:
    res = len(s)
  else:
    ...
  return res
```

Distancia de edición entre 2 strings

Notación: `s[:-1]` es s sin el último caracter; `s[-1]` es el último caracter de s

Casos recursivos:

Para calcular `ED(s,t)`, supongamos que conocemos estas distancias, porque las hemos calculado recursivamente:

- `d1 = ED(s, t[:-1])`
- `d2 = ED(s[:-1], t)`
- `d3 = ED(s[:-1], t[:-1])`

Entonces, como vimos en los ejemplos anteriores:

- si `s[-1]==t[-1]` $\Rightarrow$ `ED(s,t) = min(d1+1, d2+1, d3)`
- si `s[-1]!=t[-1]` $\Rightarrow$ `ED(s,t) = min(d1+1, d2+1)`

En el libro "Algorithms" de J. Erickson, sección 3.7, se demuestra que este algoritmo computa la distancia mínima: http://jeffe.cs.illinois.edu/teaching/algorithms/

Distancia de edición entre 2 strings

Algoritmo recursivo:

```
def ED(s, t):
    if len(s)==0:
        res = len(t)
    elif len(t)==0:
        res = len(s)
    else:
        d1 = ED(s, t[:-1])
        d2 = ED(s[:-1], t)
        if s[-1] == t[-1]:
            d3 = ED(s[:-1], t[:-1])
            res = min(d1+1, d2+1, d3)
        else:
            res = min(d1+1, d2+1)
    return res
```

Al igual que Fibonacci, tiene complejidad exponencial.

Distancia de edición entre 2 strings

Pensemos una forma de resolverla usando programación dinámica.

Vemos la idea en clase y la implementación del algoritmo queda como ejercicio (es fácil).

Supongamos que queremos calcular ED(SOL, OLAS).

	O	OL	OLA	OLAS
	Ø→Ø	Ø→O	Ø→OL	Ø→OLA	Ø→OLAS
S	S→Ø	S→O	S→OL	S→OLA	S→OLAS
SO	SO→Ø	SO→O	SO→OL	SO→OLA	SO→OLAS
SOL	SOL→Ø	SOL→O	SOL→OL	SOL→OLA	SOL→OLAS

En esta tabla, la posición (x, y) va a indicar la cantidad mínima de operaciones para transformar x en y.

		O	OL	OLA	OLAS
	Ø→Ø 0	Ø→O 1	Ø→OL 2	Ø→OLA 3	Ø→OLAS 4
S	S→Ø 1	S→O	S→OL	S→OLA	S→OLAS
SO	SO→Ø 2	SO→O	SO→OL	SO→OLA	SO→OLAS
SOL	SOL→Ø 3	SOL→O	SOL→OL	SOL→OLA	SOL→OLAS

Inicializamos la primera fila y la primera columna
(estos serían los casos base en el algoritmo recursivo)

	O	OL	OLA	OLAS
	Ø→Ø 0	Ø→O 1	Ø→OL 2	Ø→OLA 3	Ø→OLAS 4
S	S→Ø 1	S→O ?	S→OL	S→OLA	S→OLAS
SO	SO→Ø 2	SO→O	SO→OL	SO→OLA	SO→OLAS
SOL	SOL→Ø 3	SOL→O	SOL→OL	SOL→OLA	SOL→OLAS

$$S \neq O$$

$$\mathrm{ED}(S, O) = \text{mín}\Big(\mathrm{ED}(\emptyset, O) + 1 \text{ eliminación}, \quad \mathrm{ED}(S, \emptyset) + 1 \text{ inserción}\Big) = 2$$

	O	OL	OLA	OLAS
Ø→Ø 0	Ø→O 1	Ø→OL 2	Ø→OLA 3	Ø→OLAS 4
S S→Ø 1	S→O 2	S→OL	S→OLA	S→OLAS
SO SO→Ø 2	SO→O ?	SO→OL	SO→OLA	SO→OLAS
SOL SOL→Ø 3	SOL→O	SOL→OL	SOL→OLA	SOL→OLAS

$$\mathrm{O} = \mathrm{O}$$

$$\mathrm{ED}(\mathrm{SO},\ \mathrm{O}) = \text{mín}\left(\begin{array}{ll} \mathrm{ED}(\mathrm{S},\ \mathrm{O}) & +\ 1\ \text{eliminación}, \\ \mathrm{ED}(\mathrm{SO},\ \emptyset) & +\ 1\ \text{inserción}, \\ \mathrm{ED}(\mathrm{S},\ \emptyset) & \end{array}\right) = 1$$

	O	OL	OLA	OLAS
	Ø→Ø 0	Ø→O 1	Ø→OL 2	Ø→OLA 3	Ø→OLAS 4
S	S→Ø 1	S→O 2	S→OL ?	S→OLA	S→OLAS
SO	SO→Ø 2	SO→O 1	SO→OL	SO→OLA	SO→OLAS
SOL	SOL→Ø 3	SOL→O	SOL→OL	SOL→OLA	SOL→OLAS

$$\mathrm{S} \neq \mathrm{L}$$

$$\mathrm{ED}(\mathrm{S},\ \mathrm{OL}) = \text{mín}\Big(\ \mathrm{ED}(\emptyset,\ \mathrm{OL}) + 1 \text{ eliminación},\ \mathrm{ED}(\mathrm{S},\ \mathrm{O}) + 1 \text{ inserción}\ \Big) = 3$$

	O	OL	OLA	OLAS
	Ø→Ø 0	Ø→O 1	Ø→OL 2	Ø→OLA 3	Ø→OLAS 4
S	S→Ø 1	S→O 2	S→OL 3	S→OLA 4	S→OLAS 3
SO	SO→Ø 2	SO→O 1	SO→OL 2	SO→OLA 3	SO→OLAS 4
SOL	SOL→Ø 3	SOL→O 2	SOL→OL 1	SOL→OLA 2	SOL→OLAS 3

Al terminar de llenar la tabla, la solución queda en la posición inferior derecha.

Tarea: Escribir el algoritmo e implementarlo en Python.

Programación Dinámica

Objetivo: Evitar la repetición de cómputos.

Idea: Ir almacenando en una estructura de datos (p.ej., una lista, una matriz) resultados parciales que puedan ser necesarios más adelante.

Aplicaciones:

- Camino más corto entre dos nodos de un grafo (Algoritmo de Dijkstra; no confundir con el Problema del Viajante de Comercio)
- Multiplicación de cadenas de matrices
- Algoritmo de Viterbi en Hidden Markov Models (Machine Learning)
- Dynamic Time Warping, para computar la distancia global entre dos series temporales
- Método Value Iteration en Procesos de Decisión de Markov (Reinforcement Learning)

Repaso de la clase de hoy

- Cuidado con la recursión: a veces puede llevarnos a algoritmos muy ineficientes.
- Técnica de programación dinámica
- Ejemplos: Fibonacci, distancia de edición.

Con lo visto hoy, ya pueden resolver la sección 3 de la guía de ejercicios.