Programación dinámica

Brian Bokser

Objetivos

- Ver cómo encarar un problema de dinámica complejo.
- Foco en ideas.
- Entender patrones que se repiten en la técnica.

Ejercicio
CCC

Una cadena de caracteres correcta (CCC) es una cadena s que cumple una de las siguientes condiciones:

- Es nula ($\|c\| = 0$);
- Es de la forma “(c)”, “[c]” o “{c}”, donde “c” es una CCC;
- Es la concatenación de dos CCC

Ejemplos: “”, “([]{})” y “()[]{}”
No-ejemplos: “][”, “(((”, “{]” y “(” .

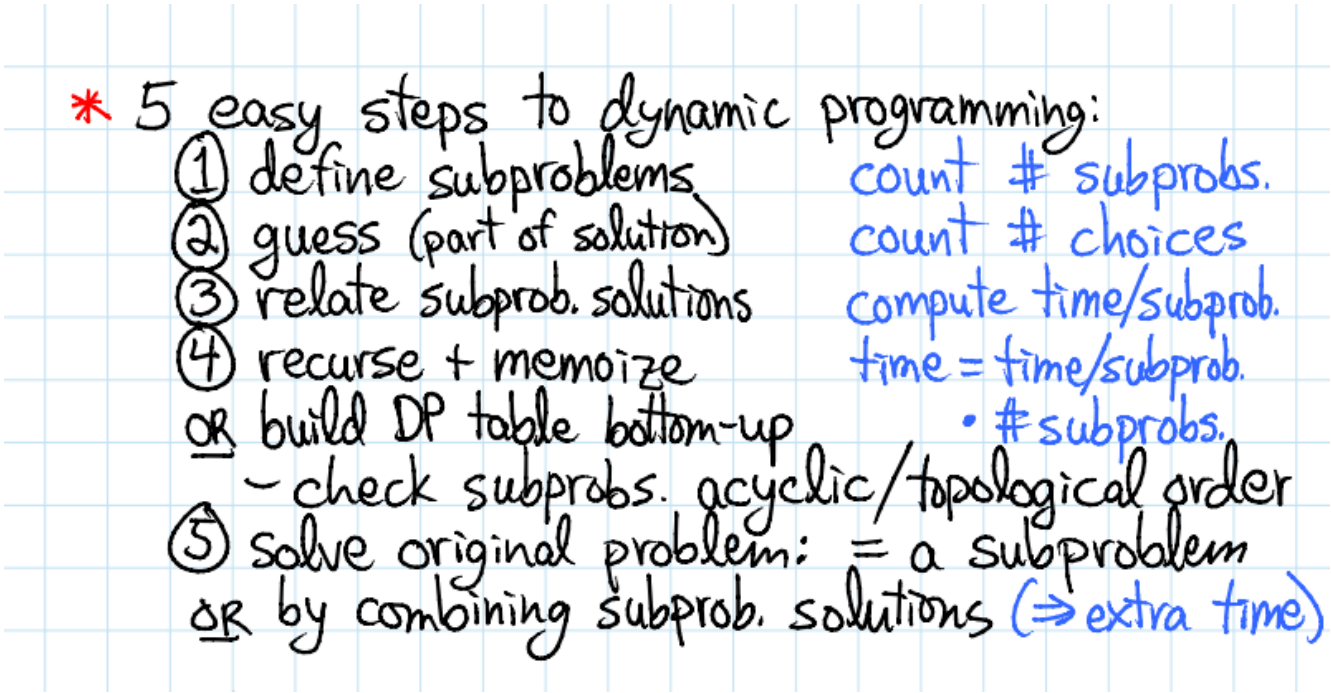
Ejercicio
CCC

Dada una cadena de caracteres de longitud par formada únicamente por los caracteres contenidos en "()[]{}", indicar la mínima cantidad de caracteres que es necesario modificar para transformarla en una CCC. Mostrar que el algoritmo propuesto es correcto y determinar su complejidad.
Ejemplos: "" $\rightarrow$ 0 "][" $\rightarrow$ 2 "(((()" $\rightarrow$ 1

Algunas propiedades

¿Siempre se puede convertir la cadena a una CCC?

5 Pasos para una Dinámica (más o menos)



De Eric Demaine [1]

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**
f(i,j) := "La mínima forma de convertir la subcadena [i..j) en CCC"

$0 \leq i \leq j \leq n$. Indexamos desde 0.

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**
f(i,j) := "La mínima forma de convertir la subcadena [i..j) en CCC"

$0 \leq i \leq j \leq n$. Indexamos desde 0.

¡Idea Importante!

¿Puedo resolver el problema original con estos subproblemas?

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**
f(i,j) := "La mínima forma de convertir la subcadena [i..j) en CCC"

$0 \leq i \leq j \leq n$. Indexamos desde 0.

¡Idea Importante!

¿Puedo resolver el problema original con estos subproblemas?
Si! f(0, n). A veces no es trivial.

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**
f(i,j) := "La mínima forma de convertir la subcadena [i..j) en CCC"

$0 \leq i \leq j \leq n$. Indexamos desde 0.

¡Idea Importante!

¿Puedo resolver el problema original con estos subproblemas?
Si! f(0, n). A veces no es trivial.

Si la respuesta es no, ¡Buscar otro subproblema!

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**
f(i,j) := "La mínima forma de convertir la subcadena [i..j) en CCC"

$0 \leq i \leq j \leq n$. Indexamos desde 0.

¡Idea Importante!

¿Puedo resolver el problema original con estos subproblemas?
Si! f(0, n). A veces no es trivial.

Si la respuesta es no, ¡Buscar otro subproblema!

¿Cuantos subproblemas hay?

Subproblemas

¿Qué subproblema podemos plantear? Definamoslo **en palabras**
f(i,j) := "La mínima forma de convertir la subcadena [i..j) en CCC"

$0 \leq i \leq j \leq n$. Indexamos desde 0.

¡Idea Importante!

¿Puedo resolver el problema original con estos subproblemas?
Si! f(0, n). A veces no es trivial.

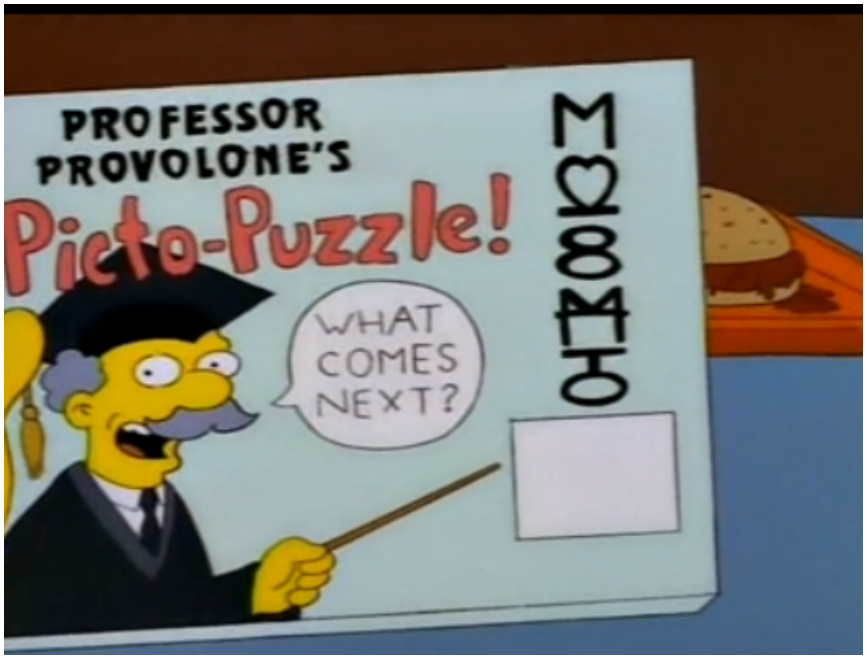
Si la respuesta es no, ¡Buscar otro subproblema!

¿Cuantos subproblemas hay? $\mathcal{O}$(n**2)

¡Adivina adivinador!

¡Idea Importante!

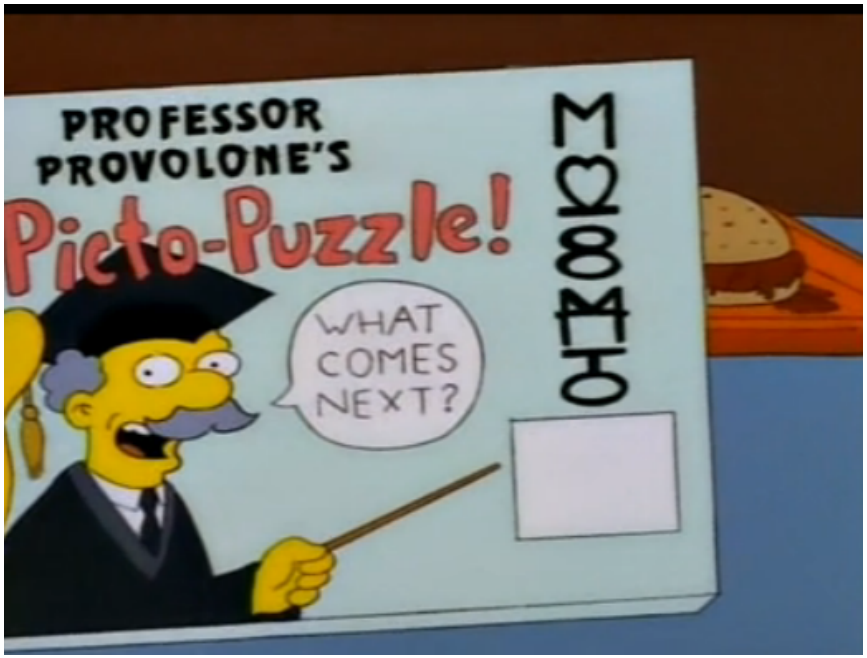
Tenemos que "adivinar" una parte de la estructura óptima. ¡Probamos **todas** *las opciones!*



¡Adivina adivinador!

¡Idea Importante!

Tenemos que "adivinar" una parte de la estructura óptima. ¡Probamos **todas** *las opciones!*

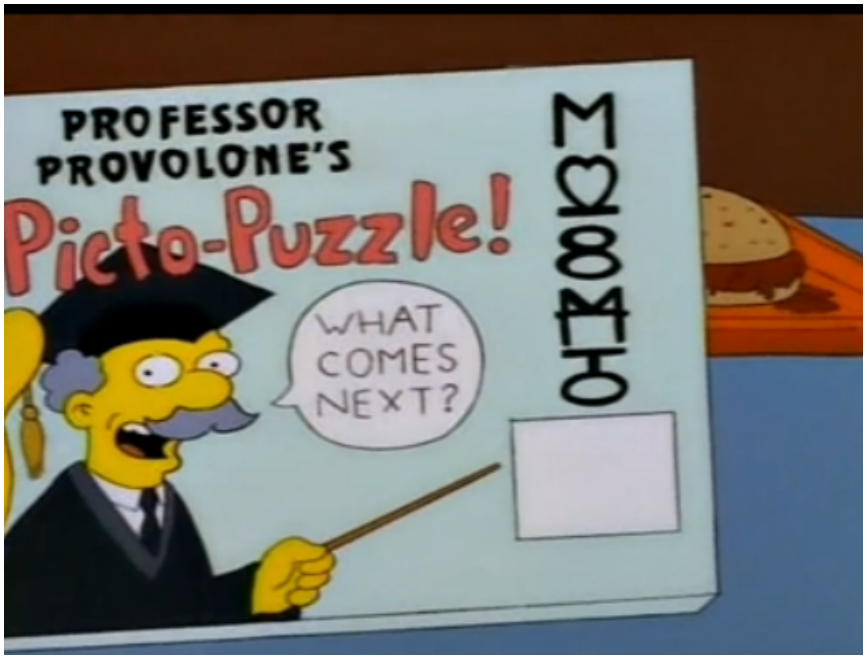


¿Que adivinamos?

¡Adivina adivinador!

¡Idea Importante!

Tenemos que "adivinar" una parte de la estructura óptima. ¡Probamos **todas** *las opciones!*



¿Que adivinamos? Cómo se conforma la CCC óptima.

- Parenteseado (c)
- Llaves {c}
- Corchetes [c]
- Concatenación

Formulación recursiva: Parenteseado

Ahora queremos **relacionar** los subproblemas. Usamos la "adivinanza"

¿Cómo calculo f(i,j) si es un parenteseado?

Formulación recursiva: Parenteseado

Ahora queremos **relacionar** los subproblemas. Usamos la "adivinanza"

¿Cómo calculo f(i,j) si es un parenteseado? f(i+1, j-1) + costo de convertir primer y último caracter a paréntesis.

Formulación recursiva: Parenteseado

Ahora queremos **relacionar** los subproblemas. Usamos la "adivinanza"

¿Cómo calculo f(i,j) si es un parenteseado? f(i+1, j-1) + costo de

convertir primer y último caracter a paréntesis.

Pero si tenemos '{' y ']', seguro que nos conviene convertir todo a llaves o corchetes, aprovechando lo que ya tenemos.

Formulación recursiva: Parenteseado

Ahora queremos **relacionar** los subproblemas. Usamos la "adivinanza"

¿Cómo calculo f(i,j) si es un parenteseado? f(i+1, j-1) + costo de

convertir primer y último caracter a paréntesis.

Pero si tenemos '{' y ']', seguro que nos conviene convertir todo a llaves o corchetes, aprovechando lo que ya tenemos.
g(i, j) := Costo de emparentar el primero y último caracter.
$g(i, j) \in \{0, 1, 2\}$ Se puede implementar con una tablita de 6×6.

Formulación recursiva: Parenteseado

Ahora queremos **relacionar** los subproblemas. Usamos la "adivinanza"

¿Cómo calculo f(i,j) si es un parenteseado? f(i+1, j-1) + costo de

convertir primer y último caracter a paréntesis.

Pero si tenemos '{' y ']', seguro que nos conviene convertir todo a llaves o corchetes, aprovechando lo que ya tenemos.
g(i, j) := Costo de emparentar el primero y último caracter.
$g(i, j) \in \{0, 1, 2\}$ Se puede implementar con una tablita de 6×6.

Costo de convertir a (c), [c], {c}: f(i+1, j-1) + g(i, j)

Formulación recursiva: Concatenación

¿Cómo calculo f(i, j) si sabemos que es una concatenación?

Formulación recursiva: Concatenación

¿Cómo calculo f(i, j) si sabemos que es una concatenación?

¡Pruebo todas las formas de concatenar!

$[i, j) = [i,k) + [k, j)$

$\min\{ f(i, k) + f(k, j) \text{ Para todo } k \}$

Formulación recursiva: Todo

$f(i, j) = 0$ si $i \geq j$

$$f(i,j) = min \begin{cases} f(i+1, j-1) + g(i,j) \\ f(i,k) + f(k,j) \forall k \in [i+2..j-2] \end{cases}$$

Top-down

```
function minimosCambios(i, j)
    if i ≥ j then
        return 0
    end if
    if dp[i,j] ≠ -1 then
        return dp[i,j]
    end if
    res ← minimosCambios(i+1, j-1) + g(i, j)
    for k ← i+2 to j-2 step 2 do
        res ← min(res, minimosCambios(i, k) + minimosCambios(k, j)))
    end for
    dp[i, j] ← res
    return res
end function
```

¿Complejidad?

¿Complejidad?

¡Idea Importante!

Total $=$ *#Subproblemas* $\times$ *Costo por subproblema*
Tomamos la recursión como $\mathcal{O}(1)$

Costo total $= \mathcal{O}(n^2) \times \mathcal{O}(\text{n}) = \mathcal{O}(n^3)$

Correctitud

Tenemos que mencionar que:

- Efectivamente resolvimos el problema. Por definición f(0, n) = problema original.
- Cálculo de f correcto. Porque miro todos los casos.
- Cada caso lo resuelvo correctamente.

Bottom-up

La implementación top-down es típicamente más clara, pero la bottom-up podría ser más eficiente (por ejemplo a nivel memoria) .

¡Idea Importante!

Cuando implementamos Bottom-Up tenemos que ocuparnos del orden de las **dependencias**.

Formalmente, seguimos un orden topológico en el DAG de subproblemas. En la materia no hacemos foco en esto.

Bottom-Up

- f(i, j) depende de subcadenas con i mayor y j menor. Decrezco i, incremento j.

```
for i ← n to 0 step -1 do
    for j ← i to n do
        dp[i,j] ← ...
    end for
end for
```

  Notar que se puede invertir el orden de los for haciendo una leve modificación.

- f(i, j) depende de subcadenas de menor tamaño. Solo subcadenas de tamaño par.

```
for t ← 0 to n step 2 do
    for i ← 0 to n-t do
        dp[i, i+t] ← ...
    end for
end for
```

Preguntas

¿Preguntas?

Biblografía + Extras

 Erik Demaine, and Srinivas Devadas. Introduction to Algorithms. Dynamic Programming II. http://bit.ly/2esOte7

Ejercicio 3.11: Minimum edit distance (Ver DP III)

Ejercicio 3.9: Operaciones. Para pensar, ¿Por qué no hay problemas de dependencias?