Backtracking

Departamento de Computación, FCEyN, UBA

27 de Agosto de 2025

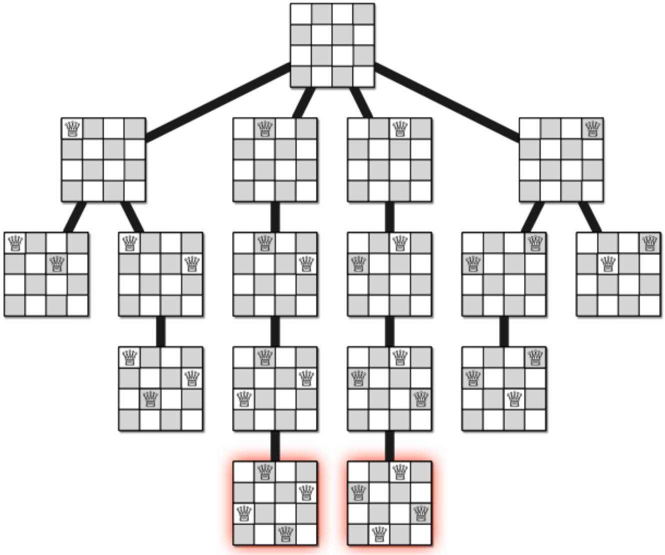
Plan para hoy

- Dar un brevísimo repaso de backtracking (5').
- Hacer 4 o 5 ejercicios de backtracking.
- Introducción a algunas demostraciones.
- Presentación del Taller $\#1$.

Backtracking

- ¿Qué es un algoritmo de fuerza bruta?
- ¿Qué es un algoritmo de backtracking?
- ¿Soluciones candidatas? ¿Soluciones parciales?
- ¿Podas por factibilidad? ¿Poda por optimalidad?

Árbol de recursión



Backtracking

Backtracking

¿Qué es un algoritmo de backtracking?

- ¿Qué es un algoritmo de fuerza bruta?
- ¿Qué es un algoritmo de backtracking?

Ejercicio 1 - Palabras en cadena

Dada una cadena de letras sin espacios o puntos queremos analizar si se puede subdividir de forma de obtener palabras. Suponiendo que se tiene una función $palabra\colon [a, z] \to bool$ que verifica si una cadena de letras es una palabra.

1. Dar una función recursiva que resuelva el problema.
2. Calcular una cota superior para la complejidad.
3. Demostrar que el algoritmo es correcto.

Ejercicio 1 - Palabras en cadena - frec

$$separar(S) = \begin{cases} True & \text{si } |S| = 0 \\ \bigvee_{i=1}^{|S|} (palabra(S[:i]) \wedge separar(S[i:])) & \text{si } |S| > 0 \end{cases}$$

Ejercicio 1 - Palabras en cadena - Predicado

$P(n) =$ **Para toda cadena** S **de tamaño** n **vale que** *separar*(S) **nos dice si hay una separación válida o no.**

Break

Ejercicio 2 - Árboles binarios de búsqueda óptimos

Dado un conjunto de elementos de $[n] = \{1, \ldots, n\}$, y una función $f : [n] \to \mathbb{N}$ que nos da la frecuencia de acceso a dichos elementos, decimos que A es un árbol binario de busqueda óptimo si este minimiza el costo de todos los accesos dados por f.

1. Escribir una función recursiva que devuelva el costo de acceder a todos los elementos de un AB dado usando f.
2. Escribir un algoritmo de backtracking que encuentre el AB óptimo para un f dado.
3. Dar una cota superior para la complejidad (Ayuda: pasar de la función recursiva a una recurrencia que solo dependa del tamaño de la entrada).
4. Probar que el algoritmo es correcto.

Ejercicio 2 - Árboles binarios de búsqueda óptimos - frec

$$AO(i,j) = \begin{cases} 0 & \text{si } i > j \\ \displaystyle\sum_{r=i}^{j} f(r) + \min_{i \leq r \leq j} AO(i, r-1) + AO(r+1, j) & \text{si no} \end{cases}$$

Break

Ejercicio 3 - Dobra

Dobra se encuentra con muchas palabras en su vida, como es una persona particular la mayoría de estas no le gustan. Para compensar empezó a inventar palabras más agradables. Dobra crea palabras nuevas escribiendo una cadena de caracteres que considera buena, luego borra los caracteres que peor le caen y los reemplaza con _. Luego para mejorar su vida intenta reemplazar estos guiones bajos con letras más aceptables intentando crear palabras más lindas. Dobra considera una palabra como buena si no contiene 3 vocales consecutivas, 3 consonantes consecutivas y al menos contiene una E. Dobra nos pide conseguir todas las posibles palabras válidas que se pueden armar a partir de una cadena con comodines.

1. Mostrar alguna solución candidata posible y alguna solución parcial.
2. Proponer una función recursiva y estimar su complejidad. [*]
3. Probar que la función o programa es correcto.
4. Proponer al menos una poda por factibilidad.
5. Si b) no tiene una cota superior $O(3^n)$ para la complejidad, analizar el caso donde se separa la recursión en tener o no una letra E y ver si mejora la misma. [¶]

---

[*] Asumir que se tiene una función *verificar* que toma una cadena y devuelve *True* si es como Dobra quiere o *False* en caso contrario.

[¶] La mejor complejidad que conocemos es $O(n2^n)$

Ejercicio 3 - Dobra - V1

$$Dobra(S, i) = \begin{cases} verificar(S) & \text{si } i = 0 \\ Dobra(S, i-1) & \text{si } S[i] \neq \_ \\ \sum\limits_{c \leftarrow \mathrm{ABC}} Dobra(S[i] \leftarrow c, i-1) & \text{si } |S| \neq i \wedge S[i] = \_ \end{cases}$$

Ejercicio 3 - Dobra - V2

$$Dobra(S, i, hayE)$$

$$= \begin{cases} 1 & \text{si } |S| = i \wedge hayE \\ Dobra(S, i+1, hayE \vee S[i] =' E') & \text{si } S[i] \neq \_ \wedge \text{es una combinación válida} \\ 4Dobra(S \leftarrow' A', i+1, hayE) & \\ \quad +Dobra(S[i] \leftarrow' E', i+1, True) & \text{si } S[i] = \_ \wedge \text{solo una vocal es válida} \\ 21Dobra(S[i] \leftarrow' B', i+1, hayE) & \text{si } S[i] = \_ \wedge \text{solo consonante} \\ 4Dobra(S \leftarrow' A', i+1, hayE) & \\ \quad +21Dobra(S[i] \leftarrow' B', i+1, hayE) & \\ \quad +Dobra(S[i] \leftarrow' E', i+1, True) & \text{si } S[i] = \_ \wedge \text{admite ambas} \\ 0 & \text{caso contrario} \end{cases}$$

Break

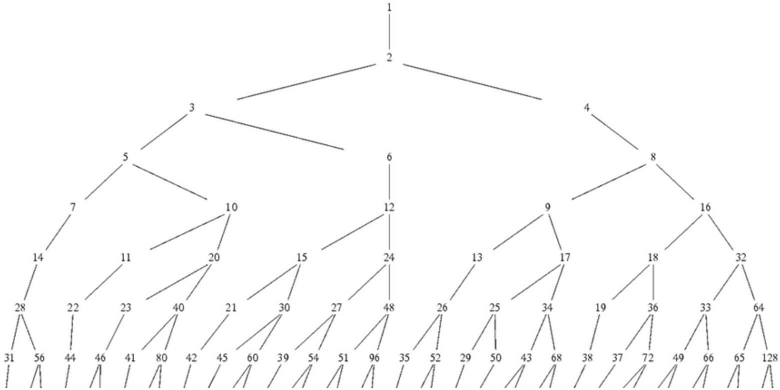
Ejercicio 4 - Cadenas de adición

Dado un entero n decimos que $C = \{x_1, \ldots, x_k\}$ es una cadena de adición si cumple lo siguiente

- $1 = x_1 < x_2 < \cdots < x_k = n$
- Para cada $2 \leq j \leq n$ existe $k_1, k_2 < j$ tal que $x_{k_1} + x_{k_2} = x_j$

1. Encontrar un algoritmo de backtracking que encuentre, si existe, la cadena de adición de longitud mínima.
2. Dar alguna poda por factibilidad/optimalidad para el algoritmo anterior.

Ejercicio 4 - Cadenas de adición



Fin

Repasemos lo que vimos hoy:

- Dimos una idea de que es un algoritmo de backtracking, que son soluciones fáctibles y óptimas.
- Hicimos varios ejercicios donde mostramos como analizar la complejidad.

Preguntas??

Taller #1

Tres ejercicios D&C y Backtracking

- https://codeforces.com/group/yuAAIJ8c1R/contest/631549/problem/A
- https://codeforces.com/group/yuAAIJ8c1R/contest/631549/problem/B
- https://codeforces.com/group/yuAAIJ8c1R/contest/631549/problem/C

Pueden hacer cuantos intentos quieran!!
No entreguen el último día, los torneos pueden hacer que los jueces anden más lento.