Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Abril 2017

Trabajo Práctico 1

Alumno	LU	Correo electrónico
Seijo, Jonathan Adrián	592/15	jon.seijo@gmail.com

Índice

1. Introducción

1.1. Explicación del problema

Dada una secuencia A de números, se quieren pintar cada uno de ellos con rojo, azul o dejarlos sin pintar. Una aclaración importante es que los elementos de A no pueden modificarse, ni tampoco cambiarse su orden inicial. Lo unico que puede hacerse con ellos es colorearlos (o no).

Para que una secuencia de colores se considere **válida** es necesario que se cumplan ciertas condiciones:

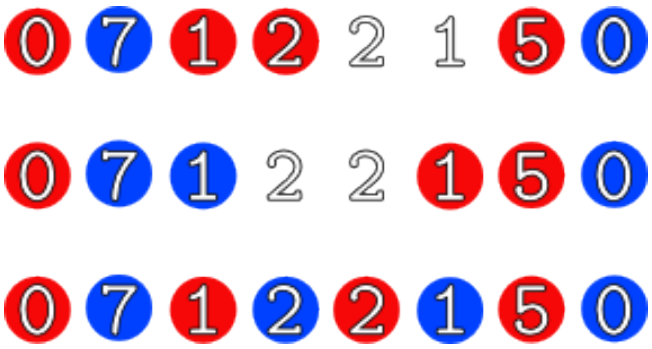
1. Todos los elementos de color rojo están ordenados por valor de forma estrictamente creciente
2. Todos los elementos de color azul están ordenados por valor de forma estrictamente decreciente

(Estrictamente significa que no hay numeros consecutivos iguales)

Las secuencias de colores válidas pueden tener diferentes cantidades de elementos sin pintar. El objetivo del problema es encontrar la **mínima cantidad de elementos sin pintar** de todas las secuencias válidas que pueden formarse a partir de A.

1.2. Ejemplo

Supongamos que $A = [0, 7, 1, 2, 2, 1, 5, 0]$. Veamos *algunas* de las posibles secuencias de colores válidas:



Consideremos los colores del tercer caso para ver que es una secuencia válida.

1. **Rojos:** [0, 1, 2, 5] (estrictamente crecientes)
2. **Azules:** [7, 2, 1, 0] (estrictamente decrecientes)

Podemos ver que diferentes formas de pintar de rojo y azul nos obligan a dejar algunos elementos sin pintar para que la secuencia sea válida. En el caso de este ejemplo la **mínima cantidad de elementos sin pintar** que puede obtenerse de A es **0**, como puede verse en la tercer combinación.

2. Backtracking

2.1. Solución

Llamo A a la secuencia de números que quiero pintar, y n a la cantidad de elementos en A. De todas las secuencias válidas de colores que puedo formar quiero saber cual es la mínima cantidad de elementos que puedo dejar sin pintar.

Una forma natural de pensar la solución es la siguiente: genero todas las formas de pintar posibles, y veo cual es el mínimo sin pintar que puede usarse para las secuencias que son válidas. Esa es la idea central detrás de ambos algoritmos de backtracking. Veamos entonces una posible implementación, la forma *naive*.

El primer elemento puede ser Rojo, Azul o Ninguno. Dado el color del primero, el segundo elemento puede también ser Rojo, Azul o Ninguno. Fijados el primero y el segundo, el tercero puede ser tomar cualquiera de las tres posibilidades, y así siguiendo.

Una vez fijos los colores de los n elementos, reviso si la secuencia de colores que se formó es válida. (Esto es, que los elementos rojos estén ordenados crecientemente y los azules decrecientemente, ambos de forma estricta).

Si la secuencia formada era válida, entonces cuento la cantidad de elementos sin pintar, y devuelvo ese número. La respuesta final es se consigue tomando el mínimo de todos los mínimos.

Como detalle de implementación, en caso de que la secuencia formada no sea válida, devuelvo un valor infinito para que no afecte al valor mínimo solución. Ésta siempre existe porque no pintar ningún elemento de ningún color es una solucion válida **finita**

2.2. Pseudocodigo

```
procedure BACKTRACK(secuencia(Colores) colores, int actual)
    if actual = n then
        if EsValido(colores) then
            return CantSinPintar(colores)
        else
            return ∞
    else
        colores[actual] ← Rojo
        minimoConRojo ← backtrack(colores, actual + 1)

        colores[actual] ← Azul
        minimoConAzul ← backtrack(colores, actual + 1)

        colores[actual] ← Ninguno
        minimoSinPintar ← backtrack(colores, actual + 1)

        return Min(minimoConRojo, minimoConAzul, minimoSinPintar)
```

Auxiliares:

procedure EsValida(secuencia(Colores) *colores*)
  bool $rojoValido \leftarrow$ EsCreciente(DameRojos(*colores*)) ▷ $O(n)$
  bool $azulValido \leftarrow$ EsDecreciente(DameAzules(*colores*)) ▷ $O(n)$
  return $(rojoValido \wedge azulValido)$

procedure CantSinPintar(secuencia(Colores) *colores*)
  return Tamaño(DameSinPintar(*colores*)) ▷ $O(n)$

2.3. Complejidad

Explicar por qué es $O(n * 3^n)$ Revisar bien pero creo que si, puedo demostrarlo haciendo la cuenta

en cada nivel son 3^k nodos, hay $n-1$ niveles y una hoja cuesta n. Entonces queda la sumatoria desde 0 hasta $n-1$ de $3^i * n = O(n * 3^n)$

3. Programación Dinámica