examenfinal2016-06AEDII.pdf

Anónimo

Algoritmos y Estructuras de Datos II

2º Grado en Ingeniería Informática

Facultad de Informática
Universidad de Murcia

RESOLUCIÓN EXAMEN AED2 JUNIO 2016. 1/1

NOTA: lo he resuelto explorando muchas opciones, en particular en 2, 5 y 6. No necesario llegar a tanto!

① a) Esta función tiene diferentes casos según el contenido de la entrada (array v). El mejor es cuando el array ya está ordenado, no se entra al mientras. El peor cuando v está inversamente ordenado, se entra siempre:

$$t_m(n) = 2 + \sum_{i=1}^{n} 6 = 2 + 6n \rightarrow t(n) \in \Omega(n).$$

$$t_M(n) = 2 + \sum_{i=1}^{n} \left(6 + \sum_{j=1}^{i-1} 3 + 1\right) = 2 + 7n + \sum_{i=1}^{n} 3(i-1)$$

$$= 2 + 7n - 3n + \frac{n(n+1)}{2} \cdot 3 = 2 + 4n + \frac{3}{2}n^2 + \frac{3}{2}n$$

$$= \frac{3}{2}n^2 + \frac{11}{2}n + 2 \longrightarrow t(n) \in O(n^2)$$

→ $t(n)$ no tiene orden exacto (Θ), ya que el Θ de sus casos peor y mejor difiere.

b) No se indica la base del logaritmo. Elijo 4. Esto no afecta al orden. Cambio de variable: $n = 4^k$, $k = \log_4 n$.

$t'(k) - 2t'(k-1) = k$ (lineal NO homogénea)

EC. CARACTERÍSTICA: $(x-2)(x-1)^2 = 0 \Rightarrow$ SOLS: $\begin{cases} x = 2 \\ x = 1 \\ x = 1 \end{cases}$

SOLUCIÓN GENÉRICA: $t'(k) = c_1 \cdot 2^k + c_2 \cdot 1^k + c_3 \cdot k \cdot 1^k$

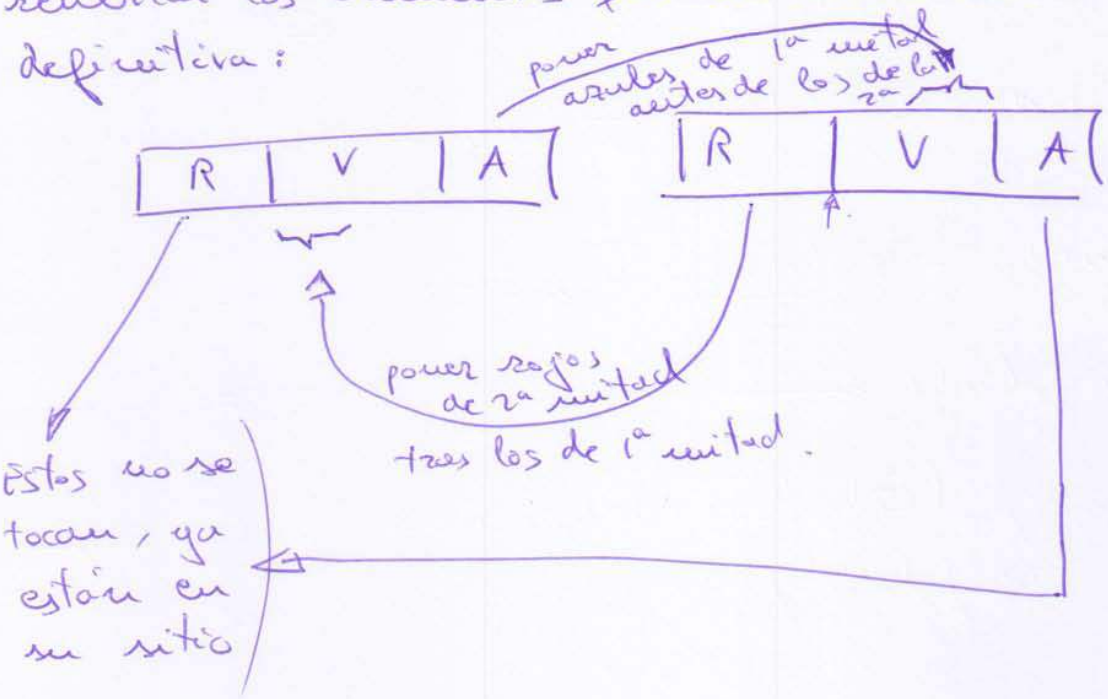
EN n... $t(n) = c_1 \cdot 2^{\log_4 n} + c_2 + c_3 \log_4 n$.

$= c_1 \cdot n^{1/2} + c_2 + c_3 \cdot \log_4 n \Rightarrow t(n) \in \Theta(\sqrt{n})$

CASOS BASE: $n = 1, 4, 16$

(2) Una forma directa de resolverlo es aplicar un quicksort, sin más que asumir una relación de orden entre los tres colores: $R < V < A$.
También podemos verlo como que $R=1$, $V=2$, $A=3$.
El orden del tp sería $n \cdot \log n$, con $t_m(n) = n^2$.

Otra forma (PARA NOTA!): dividir por la mitad y al combinar reubicar los elementos que no estén aún en su posición definitiva:



⇒ Para ahorrar tiempo, los subproblemas devuelven índices que indican dónde está el ~~primer verde~~ último rojo y 1er azul.
Si son = indican que no hay elementos de ese color

ESQUEMA

```
D.y.V (i, j) : (v, a).
    SI pequeño (i, j) DEVOLVER soldirecta (i, j).
    SI NO.
        k = dividir (i, j)
        (v1, a1) = D.y.V (i, k)
        (v2, a2) = D.y.V (k+1, j)
        devolver combinar (i, k, j, v1, a1, v2, a2)
    FIN SI
```

((2, cont.)) FUNCIONES

* pequeño (i, j) : devolver i == j // 1 elemento
* sol directa (i, j) : devolver SEGÚN v[i] : { r → (0,0); v → (0,0); a → (0,1) }
* dividir (i, j) : devolver $\frac{i+j}{2}$.

* combinar (i, k, j, r1, a1, r2, a2) ("params")

SI !(hay rojo en 2 (params)) ENTONCES ult-rojo = r1.
SI NO SI !(hay que mover rojos (params))
ENTONCES ult-rojo = r2.
SI NO ult-rojo = mover-rojos (params).
FINSI
FINSI.

SI !(hay azul en 1 (params)) ENTONCES primer-azul = a2
SI NO SI !(hay que mover azules (params))
ENTONCES primer azul = a1.
SI NO primer-azul = mover-azules (params)
FIN SI
FIN SI.

2) FUNCIONES AUXILIARES DE COMBINAR

* hay rojo en 2 (params): devolver r2 > 0.
* hay que mover rojos (params): devolver r1 < K.
* mover rojos (params):

```
origen = r2.
destino = r1 + 1.
MIENTRAS origen ≥ (K+1) Y destino ≤ K.
    intercambiar (origen, destino).
    origen --
    destino ++
FIN MIENTRAS.
devolver destino.
```

* hay azul en 1 (params): devolver a1 > 0.
* hay que mover azules (params): devolver a2 > K+1.
* mover azules (params)

```
origen = a1
destino = a2 - 1.
MIENTRAS origen ≤ K Y destino ≥ K+1.
    intercambiar (origen, destino).
    origen ++
    destino --.
FIN MIENTRAS
devolver destino.
```

NOTA: una vez intercambiados rojos y azules, los verdes ya están todos en su sitio.

⇒ el orden es n. log n en todos los casos.

(3) Este problema es de tipo DAG, así que la recurrencia es bastante directa:

$$\overbrace{\text{mejorRumo}}^{MR}(i,j) = \overbrace{\text{tablero}}^{T}(i,j) + MAX \begin{cases} MR(i-1,j) \\ MR(i-1,j-1) \\ MR(i,j-1) \end{cases}$$

~~Asumo que~~

$MR(i,j) = -\infty$ para $i<1$ o $j<1$.

$MR(i,j) = T(i,j) = -\infty$ para casillas con cactus

CASO BASE: $M(1,1) = T(1,1)$.

TABLA: 2D, $\begin{cases} 1 \text{ — } N \\ 1 \text{ — } N \end{cases}$

Rellenar por filas (arriba → abajo) de izq. a dcha:

```
FOR i = 1 — N.
    FOR j = 1 — N.
        TABLA[i,j] = T[i,j] + MAX( TABLA(i-1,j),
                                   TABLA(i-1,j-1),
                                   TABLA(i,j-1) )
        // donde los accesos a TABLA fuera de rango devuelven -∞
```

SOLUCIÓN ÓPTIMA: en TABLA[N,N].

Para reconstruir, array solución de pares de coordenadas longitud entre N y 2N. Inicializar con casilla (N,N). En cada casilla (i,j) empezando en (N,N), ver desde qué casilla se llegó (la que sumador sea TABLA(i,j) − T[i,j]), añadirla a la solución, y actualizar (i,j) = esa casilla hasta (i,j) = (1,1).

Descansito &Coins

Compra más barato eso que tanto querías o vende las cosas que te dejó tu ex.

No llores, factura

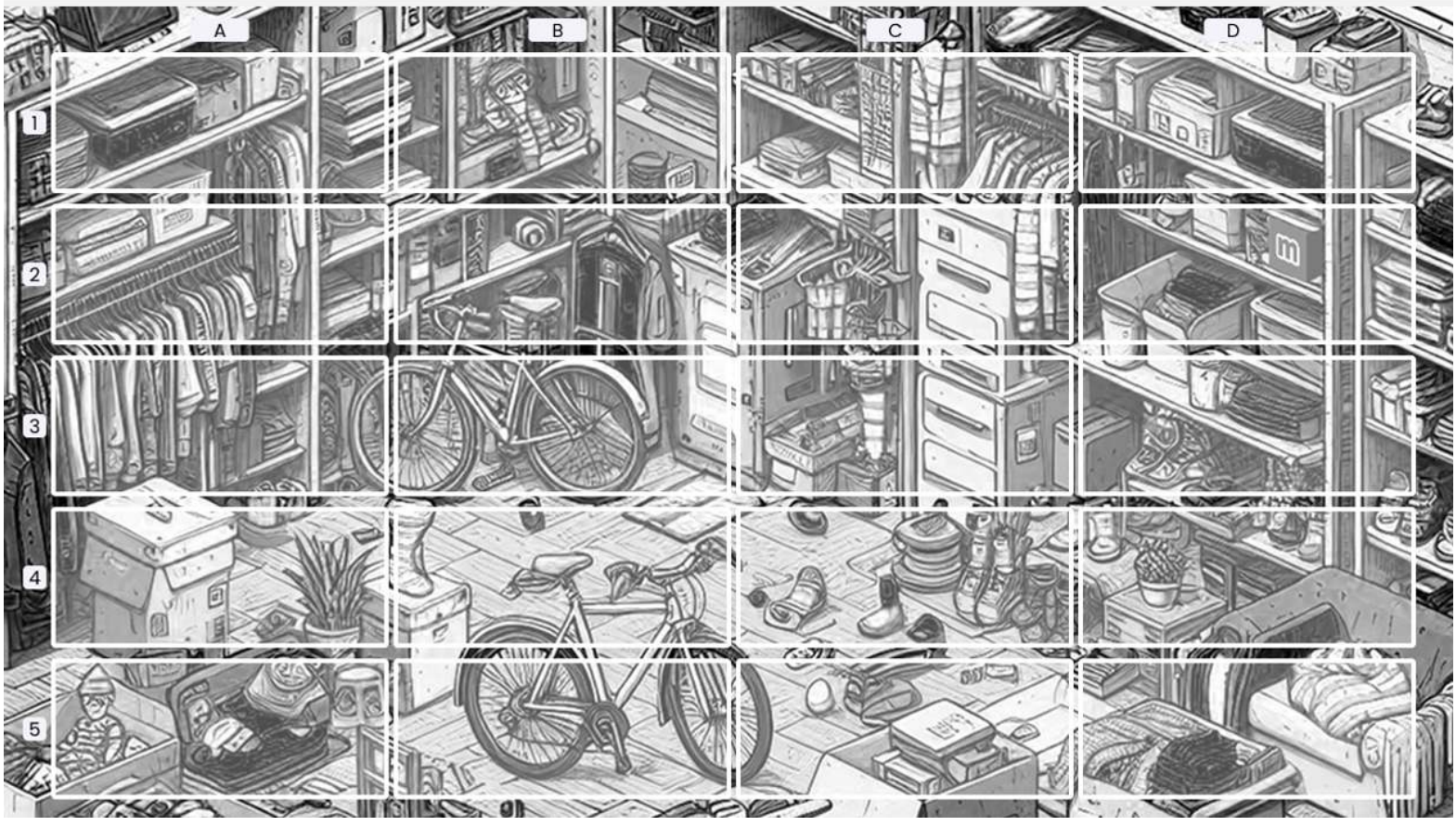
Encuentra la caja de Milanuncios en el trasteroy gana
5 coins para descargas sin publi.

Juega Ahora

REGLAS

1. Encuentra el objeto oculto en el anuncio dentro de tu apunte.
2. Escanea el QR para acceder al juego en Wuolah.
3. Introduce la coordenada donde se esconde el producto.
4. Gana tu recompensa 🎉

Fácil | 5



(4) VORACES

La función selección elige, en cada paso, la casilla siguiente con mayor valor. Si el conjunto de candidatos tiene N elementos la 1ª vez y 2 o 3 los demás. Siempre hay solución. Siempre N pasos.

No hay "factible", selección garantiza factibilidad

```
SOL = ∅; nivel = 1;
candidatos = {1, ..., N}.
MIENTRAS ! SOLUCIÓN(nivel)
    x = seleccionar (candidatos).
    SOL(nivel) = x;
    candidatos = generar candidatos (nivel, x).
FIN MIENTRAS.
```

FUNCIONES

* solución (nivel): devolver nivel == N.
* seleccionar (candidatos): devolver índice del max(cand)
* generar candidatos (nivel, x) // no considera nivel 1.

SEGÚN x devolver
- 1 → {1, 2}
- N → {N-1, N}
- otro caso → {x-1, x, x+1}

(5) BACKTRACKING

Árbol "k-ario", donde K = N para el primer nivel y 2ó3 para el resto. Tupla = N números, cada uno indica la casilla elegida en cada fila (columna)

ESQUEMA: optimización (max).

SOLUCIONES: en hojas (si cumple hoy).

```
BACKTRACKING(N, T, var SOL).
  nivel = 1; bact = 0;
  SOA = ∅.
  VOA = -∞.
  SOL = [0, ..., 0] // valor inicial tupla sol.
  REPETIR
    GENERAR(nivel, sol, bact);
    SI SOLUCIÓN(nivel, N) Y VALOR(sol) > VOA.   (bact)
        SOA = sol; VOA = VALOR(sol).
    SINO SI CRITERIO(nivel$, sol).
        nivel = nivel + 1;
    SINO MIENTRAS NO masHermanos(nivel, sol)
        Y NIVEL > 0 Y CRITERIO2(nivel, sol)
        RETROCESO(nivel, sol, bact)
  HASTA. nivel == 0.
```

FUNCIONES

```
GENERAR (nivel, sol, bact).

   SI sol[nivel] == 0   //1er hermano.
      SI nivel == 1, sol[nivel] = 1.
      SI NO. SI sol[nivel-1] == 1, sol[nivel] = 1.
             SI NO sol[nivel] = sol[nivel-1] - 1;

   SI NO
      bact = bact - T[nivel, sol[nivel]];
      sol[nivel] ++;

   FIN SI.
   bact = bact + T[nivel, sol[nivel]];
```

```
SOLUCION (nivel, N), devolver nivel == N.
```

```
VALOR (sol), devolver bact.
```

```
CRITERIO (nivel, sol, bact)
   SI nivel == N devolver FALSO;
   cota = bact + (N - nivel + 1) . máx desde casilla s[nivel] (T, nivel
devolver cota > voa.
```

```
MASHERMANOS (nivel, sol)
   SI nivel = 1 devolver sol[nivel] < N.
   SI NO. SI sol[nivel-1] == N devolver sol[nivel] < N
          SI NO devolver sol[nivel] < sol[nivel-1]+1
          FIN SI
   FIN SI.
```

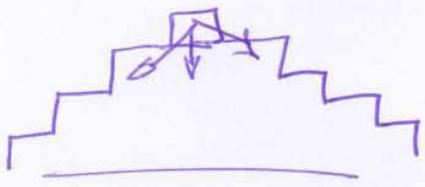
```
RETROCESOR (nivel, sol, bact)
  bact = bact - T[nivel, sinivel]];
  sinivel] = 0.
  nivel - -.
```

---

maxdesdefila (T, i, j) es una función que acota el beneficio conseguible desde la casilla (i, j) hacia abajo. Hay varias opciones, de menos a + precisa:

* (n - nivel + 1) * max (T).
  - ↳ tarda n^2 en calcular, 1 sola vez, antes del BT
  - ↳ t. cte.
* (n - nivel + 1) * maxdesdefila (T, nivel).
  - t cte.
* $\sum_{i=nivel}^{N}$ maxfila (T, i) → puede precalcularse en n^2 y almacenar en array, acceso en t. constante

* Conociendo el movimiento:



puede precalcularse, en n^1, el máximo alcanzable con el "triángulo" bajo cada casilla (i, j).

Elegiría esta opción, es bastante precisa y no añade orden al BT.

(6) R y P \10

Usaré mismo árbol y tupla solución que en BT.

NODO INICIAL { tupla, nivel = 0, bact = 0, C1, BE, CS. | → Por simplificar, quiero N nodos iniciales en vez de este: ns hijos.
NODOS INICS ∀ i=1..N { tupla = [i, 0, .., 0], nivel = 1, bact = T[1, i]. ...

SOLUCIÓN cuando nivel = N; Todos los nodos generan.

Generar hijos: para todo y hijo de x... (para nivel ≥ 2).
↳ para i = -1, 0, 1

```
y.tupla = x.tupla.
y.nivel = x.nivel + 1.
si x.tupla[x.nivel] == 1 y i == -1, BREAK
si x.tupla[x.nivel] == N y i == 1, BREAK.
y.tupla[y.nivel] = x.tupla[x.nivel] + i;
y.bact = x.bact + T[nivel, y.tupla[y.nivel]];
```

Ramificación: MB - LIFO (favorecer profundidad, sols en hojas)

PODA: C = MAX(sols, CI(nodo)). /podar i si CS(i) ≤ C

COTAS

- CS: los del BT
- CI: voraz de 4 (siempre hay sol)
- BE: $\frac{CI+CS}{2}$

Esquema (--).