



Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

E.T.S.I. Informática y de Telecomunicación, C/Periodista Daniel Saucedo Aranda s/n - 18015 - Granada (España)

UNIVERSIDAD
DE GRANADA

Grado en Ingeniería Informática Algorítmica

Curso 2021/2022. Convocatoria extraordinaria.

12 de Julio de 2022

1. (2 puntos) Calcular el orden de eficiencia en notación $O(\cdot)$ de la llamada al procedimiento $\text{proc}(v, 0, n-1)$, donde v es un vector de n números enteros.

```
int proc(const int *v, const int i, const int j) {
    if (i >= j)
        return v[i];
    else {
        int tam = j - i;
        int pt = i + (tam / 3); i+26/3
        int st = i + (tam * 2) / 3;
        int rpt = proc(v, i, pt); T(4/3)
        int rtt = proc(v, st+1, j);
        int r = rpt+rtt; L> T(4/3)
        long s = 0;
        for(int k=i;k<=j;k++)
            s+=v[k]+r-k;
        return s;
    }
}
```

$$T(n)=2T\left(\frac{n}{3}\right)+n \rightarrow \text{Easy}$$

2. (2 puntos) Para estudiar un examen de n temas disponemos de T días. Conocemos el número de días completos que necesitamos para estudiar cada uno de los temas (t_1, t_2, \dots, t_n). Se pide diseñar un algoritmo de Programación Dinámica que permita estudiarnos el máximo número posible de temas antes de que llegue la fecha del examen.
3. (2 puntos) Sea un conjunto C contenido en n actividades de un calendario. Cada actividad i tiene un instante de tiempo s_i de comienzo, y un instante de tiempo e_i donde debe haber ya finalizado. Se pide: Diseñar un algoritmo greedy que nos permita encontrar el subconjunto que contenga el máximo número de actividades compatibles. Dos actividades i, j se dicen compatibles si no se solapan en el tiempo, es decir, si $[s_i, e_i] \cap [s_j, t_j] = \emptyset$.
4. (2 puntos) Sea C un conjunto de n elementos desordenado. Deseamos construir un algoritmo Divide y Vencerás que, sin necesidad de ordenar el conjunto, nos diga qué elemento del conjunto estaría exactamente en la posición $\lfloor n/2 \rfloor$ si el conjunto estuviese ordenado (nota: $\lfloor x \rfloor$ indica parte entera de x). Por ejemplo si $n = 9$ (y $\lfloor 9/2 \rfloor = 4$) y el conjunto $(4, 7, 6, 1, 2, 10, 8, 11, 3)$ lo representamos en un array $(v[0], v[1], \dots, v[8])$, el elemento que estaría en la posición central $v[4]$ sería 6.
5. (2 puntos) Supongamos que tenemos n hombres (h_1, h_2, \dots, h_n) y n mujeres (m_1, m_2, \dots, m_n) y dos matrices $H_{n \times n}$ y $M_{n \times n}$. La fila $H[h_i, \cdot]$ contiene las preferencias del hombre h_i por las mujeres m_1, \dots, m_n , y la fila $M[m_i, \cdot]$ contiene las preferencias de la mujer m_i por los hombres h_1, \dots, h_n . Se pide: Diseñar un algoritmo de exploración en grafos que haga una asignación uno a uno entre hombres y mujeres de modo que se maximice la suma de los productos de las preferencias de ambos. Por ejemplo, si $f(m)$ es el hombre asignado a la mujer m , se desea maximizar: $\sum_{i=1}^n M[m_i, f(m_i)]H[f(m_i), m_i]$.

Duración del examen: 2 horas y 30 minutos.

NOTA: En todos los problemas se debe emplear alguna de las técnicas de diseño o análisis de algoritmos estudiadas en esta asignatura.

2. (2 puntos) Para estudiar un examen de n temas disponemos de T días. Conocemos el número de días completos que necesitamos para estudiar cada uno de los temas (t_1, t_2, \dots, t_n). Se pide diseñar un algoritmo de Programación Dinámica que permita estudiarnos el máximo número posible de temas antes de que llegue la fecha del examen.

Asumimos si un tema se estudia, que el tiempo para estudiarlo no se elige, basta que no se estudie

Vemos como objetivo inicial que se cumple el POB, sea j el tema sucesor de temas que sea solución óptima con $i+1$. Sabemos que j es el número de temas para T días. Vemos que j es una solución óptima para $T - \sum_{k=0}^i t_k$ días

Supongamos que j no es óptimo, luego existe otra combinación de temas que provoca un valor mayor dentro de ese número de días. Luego al considerar $j+1$ obtendremos una solución con un mayor número de temas.

Es posible que se haga la siguiente confusión: examen-tema.

Planteamos ahora el algoritmo:

• $d(i, j) \rightarrow$ máximo de temas posibles considerando los temas $\{0, \dots, i\}$ en j días

• $d(i-1, T) \rightarrow$ solución al problema

• $d(i, j) = 0$ si $c(i) > j$ // de 0 a i

$$d(i, j) = \begin{cases} d(i-1, j) & \text{si } T - \sum_{k=0}^{i-1} c(k) < c(i) \\ i + d(i-1, j - c(i)) & \text{si } T - \sum_{k=0}^{i-1} c(k) \geq c(i) \end{cases}$$

$$d(i, j) = \max \{ d(i-1, j), i + d(i-1, j - c(i)) \}$$

Algoritmo

Para cada i desde 0 hasta T

Para cada j desde 0 hasta i

si $c(i) > j$

$$d(i, j) = 0 // \text{Tan solo el otro caso}$$

Para cada i desde $i+1$ hasta T

Para cada j desde $j+1$ hasta T

$$d(i, j) = \max \{ d(i-1, j), \underbrace{i + d(i-1, j - c(i))}_{\text{si } j - c(i) > 0 \rightarrow -r} \}$$

3. (2 puntos) Sea un conjunto C contenido en n actividades de un calendario. Cada actividad i tiene un instante de tiempo s_i de comienzo, y un instante de tiempo e_i donde debe haber ya finalizado. Se pide:
Diseñar un algoritmo greedy que nos permita encontrar el subconjunto que contenga el máximo número de actividades compatibles. Dos actividades i, j se dicen compatibles si no se solapan en el tiempo, es decir, si $[s_i, e_i] \cap [s_j, t_j] = \emptyset$.

Esto es un greedy dado por la siguiente estructura:

- **Paquetes**: pares de enteros que representan a las actividades
- **Selcciónados**: actividades que no se solapan
- **Función selección**: Toma la actividad que acaba antes, si no se solapa con las que tengo. En caso de que sí lo haga la borra
- **Función factible**: Todas las soluciones son factibles si no hay solapamientos
- **Función solución**: comprueba que la "solución" sea un vector de pares
- **Función objetivo**: la solución es óptima si maximiza el número de actividades realizadas

Vamos a demostrar que la solución óptima

-) Primera elección. Supongamos que $\exists S$ solución que falleza a a_1 , como primera elección, siendo la actividad que termina antes. Luego será σ_1 la primera elección, pero como $a_1, s_1 < \sigma_1, s_1$ puede existir a_2 otra actividad no solapable con a_1 , tal que $a_2, e_2 < \sigma_1, e_1$. Luego $a_2, u_{a_2} < \sigma_1$, por tanto $a_2, u_{a_2}, u_{S \cup \sigma_1}$ es mejor solución.
-) Subestructuras óptimas. Sea S una solución óptima, veremos que $S \setminus a_n$ es óptima para el problema de lista de fin de actividades σ_{n+1}, \dots, e_m . Supongamos que no lo fuera, entonces $\exists S'$ solución al nuevo problema tal que $\#S' \geq \#S \setminus a_n$ (luego $\#(S' \cup \sigma_{n+1}) \geq \#S$ por tanto S' no es solución óptima. Contradicción)

La implementación, es pseudocódigo, del algoritmo se dejó al lector

5. (2 puntos) Supongamos que tenemos n hombres (h_1, h_2, \dots, h_n) y n mujeres (m_1, m_2, \dots, m_n) y dos matrices $H_{n \times n}$ y $M_{n \times n}$. La fila $H[h_i, \cdot]$ contiene las preferencias del hombre h_i por las mujeres m_1, \dots, m_n , y la fila $M[m_i, \cdot]$ contiene las preferencias de la mujer m_i por los hombres h_1, \dots, h_n . Se pide: Diseñar un algoritmo de exploración en grafos que haga una asignación uno a uno entre hombres y mujeres de modo que se maximice la suma de los productos de las preferencias de ambos. Por ejemplo, si $f(m)$ es el hombre asignado a la mujer m , se desea maximizar: $\sum_{i=1}^n M[m_i, f(m_i)]H[f(m_i), m_i]$.

No comprendo muy bien el problema

Creo que podré funcionar mejor o "greedy".

Diseñar la clase solución: vector cada par es un hombre y el valor es el índice de la mujer (y viceversa).

• Cola local sacar las veces que quedan el máximo de cada i para considerar y multiplicar.

Ojo cuando que las mujeres hubieran preferencias, luego son clarificadores.

• Cola global aplicar un greedy.

4. (2 puntos) Sea C un conjunto de n elementos desordenado. Deseamos construir un algoritmo Divide y Vencerás que, sin necesidad de ordenar el conjunto, nos diga qué elemento del conjunto estaría exactamente en la posición $\lfloor n/2 \rfloor$ si el conjunto estuviese ordenado (nota: $\lfloor x \rfloor$ indica parte entera de x). Por ejemplo si $n = 9$ (y $\lfloor 9/2 \rfloor = 4$) y el conjunto $(4, 7, 6, 1, 2, 10, 8, 11, 3)$ lo representamos en un array $(v[0], v[1], \dots, v[8])$, el elemento que estaría en la posición central $v[4]$ sería 6.

El algoritmo consistiría en realizar el proceso del Quicksort, en el peor caso ordenaríamos el vector pero en el caso promedio no
 ↓
 juega con búsqueda binaria

int Mediana (vector v , int i_u , int f_u , int i_m , int f_m);
Este es el codificado pero la idea es la misma que sección 2. V

if ($k == 0$) return -1;

if ($k == 1$) return $v[0]$;

if ($k == 2$) return $v[0]$;

else {

int med = $v[\text{mitad}]$;

Proceso (v , i_u , f_u , i_m , f_m);

if ($v[\text{mitad}] == \text{med}$) return med ;

else {

if (los dos extremos a mitad son mayores)

Mediana (v , i_u , f_u , i_m , f_m);

else if (son iguales)

Mediana (v , i_m , f_m , i_u , f_u);

else Mediana al completo

f f

{