

Grado en Informática Algorítmica

Curso 2019/2020. Convocatoria extraordinaria
21 de julio de 2020

1. (2 puntos) Calcular el orden de eficiencia en notación $O(\cdot)$ del siguiente algoritmo:

```
proc(v, 0, n-1)
```

donde v es un vector de n números enteros, y el procedimiento `proc` es:

```
int proc(v,prim,ult) {
    if (prim==ult) return v[prim];
    else if (prim+1==ult) return v[prim]+v[ult];
    else {
        mitad=(prim+ult)/2;
        izq=proc(v,prim,mitad);  $T(\frac{n}{2})$ 
        der=proc(v,mitad+1,ult);  $T(\frac{n}{4})$ 
        mi=(prim+mitad)/2;
        md=(mitad+1+ult)/2;
        ce=proc(v,mi+1,md);  $T(\frac{n}{4})$ 
        q1=proc(v,prim,mi);  $T(\frac{n}{4})$ 
        q2=proc(v,mi+1,mitad);  $T(\frac{n}{4})$ 
        q3=proc(mitad+1,md);  $T(\frac{n}{4})$ 
        q4=proc(md+1,ult);  $T(\frac{n}{4})$ 
        s=0;
        for (i=prim; i<=ult; i++)  $n^2$ 
            for (j=prim; j<=ult; j++)
                s+=v[i]+j;
        return izq+der+ce-q1-q2-q3-q4+s;
    }
}
```

$$T(n) = 2T\left(\frac{n}{2}\right) + 5T\left(\frac{n}{4}\right) + n^2$$

2. (2 puntos) Dado un tablero de tamaño $n \times n$ y una ficha, se puede mover la ficha de arriba hacia abajo del tablero de acuerdo a las siguientes reglas: estando en la casilla (i, j) , podemos movernos a las casillas $(i + 1, j - 1)$ (si $j \neq 1$), $(i + 1, j)$ y $(i + 1, j + 1)$ (si $j \neq n$). Si nos movemos a una cierta casilla (i, j) , obtenemos un beneficio de $p(i, j)$ unidades. El objetivo es encontrar la forma de colocar la ficha en la primera fila, hacer movimientos legales hasta llegar a la última fila y conseguir el máximo beneficio posible.

Diseñad un algoritmo eficiente para resolver el problema de forma óptima. Aplicadlo para resolver un tablero 5×5 con beneficios:

↳ Ingáiteu

↳ Backtracking visitando las casillas por las que pasa



UNIVERSIDAD
DE GRANADA

8	2	4	5	6
10	12	5	15	7
2	6	9	7	3
13	11	7	8	1
4	6	10	9	5

3. (2 puntos) Tenemos n objetos de pesos positivos w_1, w_2, \dots, w_n , y un número ilimitado de recipientes iguales con capacidad máxima R (siendo $w_i \leq R, \forall i$). Los objetos se deben meter en los recipientes sin partirlos, y sin superar su capacidad máxima. Se pretende usar el mínimo número de recipientes necesarios para colocar todos los objetos. Diseñad un algoritmo que resuelva el problema de forma óptima. → *Prog Dinámica*
4. (2 puntos) Se tiene un vector de números enteros v de tamaño n . Se dice que los elementos de las posiciones i y j están alterados, si se cumple que $i < j$ pero $v[i] > v[j]$. Se desea contar el número total de alteraciones que hay en el vector. Por ejemplo para el vector:

i	1	2	3	4	5	6	7	8
$v[i]$	6	1	4	2	3	8	7	5

DyV, fuerza bruta.

el número de alteraciones es 10 (pares 6-1, 6-4, 6-2, 6-3, 6-5, 4-2, 4-3, 8-7, 8-5, 7-5). Diseñad un algoritmo lo más eficiente posible para realizar esta tarea.

5. (2 puntos) En una comarca en la que hay un conjunto de pueblos (interconectados por una red de carreteras) se está planeando instalar un hospital. Se han preseleccionado tres de esos pueblos como posibles ubicaciones del hospital. El criterio final para decidir cuál de los tres pueblos será el elegido es que esté lo más cerca posible del conjunto de pueblos restante. Formalizad el problema y describid detalladamente un algoritmo lo más eficiente posible para resolverlo.

No DyV, Kruskal?

Duración del examen: 2 horas y 30 minutos.

Nota importante: En todas las preguntas excepto la primera, especificad claramente qué tipo de algoritmo se va a utilizar y cómo ese tipo de algoritmo se concreta para el problema en cuestión.

3. (2 puntos) Tenemos n objetos de pesos positivos w_1, w_2, \dots, w_n , y un número ilimitado de recipientes iguales con capacidad máxima R (siendo $w_i \leq R, \forall i$). Los objetos se deben meter en los recipientes sin partirlos, y sin superar su capacidad máxima. Se pretende usar el mínimo número de recipientes necesarios para colocar todos los objetos. Diseñad un algoritmo que resuelva el problema de forma óptima. \rightarrow Prog. Dinámica

Este problema se asimila al problema de la mochila 0/1. Dicho lo resolveremos por programación dinámica:

Las decisiones se basan en tomar o no un recipiente. Es decir, un vector nuevo confección de todos entendidos como objetos de manera que

$v(i)$ = nº de recipientes usados para resolver el problema con los primeros i objetos

$v(i-1)$ = solución al problema.

guardando
el número
de los recipientes

$v(i) = v(i-1)$ si el objeto entra en alguno de los recipientes ya usado.

$v(i) = 1 + \min \{ v(j) \mid j \in \{0, 1, \dots, i-1\} \}$ si el objeto requiere un recipiente nuevo.

Por tanto, la recursión viene dada por

$$v(i) = \min \{ v(i-1), v(i-1) + 1 \}$$

El caso base es $v(0)=1$ pues siempre necesitaremos un recipiente para el primer objeto.

Por tanto, el algoritmo sería el siguiente:

$$v[0] = 1$$

Actualiza (r, v)

para cada i en $1, \dots, n-1$

si $\text{peso}[i] \leq \text{max}(r)$

$$v(i) = v(i-1)$$

si no

$$v(i) = \min(v(i-1), v(i-1) + 1)$$

Actualiza (r, v)

fin para cada i

4. (2 puntos) Se tiene un vector de números enteros v de tamaño n . Se dice que los elementos de las posiciones i y j están alterados, si se cumple que $i < j$ pero $v[i] > v[j]$. Se desea contar el número total de alteraciones que hay en el vector. Por ejemplo para el vector:

i	1	2	3	4	5	6	7	8
$v[i]$	6	1	4	2	3	8	7	5

DyV, fuerza bruta.

el número de alteraciones es 10 (pares 6-1, 6-4, 6-2, 6-3, 6-5, 4-2, 4-3, 8-7, 8-5, 7-5). Diseñad un algoritmo lo más eficiente posible para realizar esta tarea.

Vamos a resolver el problema usando Divide y Conquer con algoritmo de la asignatura. No obstante, probablemente luego un algoritmo fuerza bruta como el siguiente sea más eficiente:

Idea principal: Buscar la relación entre el vector ordenado (\neq alteraciones) y el vector original.

int AlteracionesDyV (int v , int n , int $inicio$)

if ($n=0$) return 0;

if ($n=1$) return $v[0]>v[1] ? 1 : 0$;

else {

int alteraciones = AlteracionesDyV ($v, \frac{v.size()}{2}, 0$) + AlteracionesDyV ($v, \frac{v.size()}{2}, \frac{v.size()}{2}$)

// Recursividad

for (int i=0; i< $\frac{v.size()}{2}$; ++i)

if ($v[i] > v[i + \frac{v.size()}{2}]$)

alteraciones++;

return alteraciones;

}

Este bucle está mal puesto para cada elemento de

la izquierda de bienvenida considerar todos los de la derecha

5. (2 puntos) En una comarca en la que hay un conjunto de pueblos (interconectados por una red de carreteras) se está planeando instalar un hospital. Se han preseleccionado tres de esos pueblos como posibles ubicaciones del hospital. El criterio final para decidir cuál de los tres pueblos será el elegido es que esté lo más cerca posible del conjunto de pueblos restante. Formalizad el problema y describid detalladamente un algoritmo lo más eficiente posible para resolverlo.

↳ No puede ser Kruskal, es algo parecido pero empezando desde ese punto, Prim o Ford-Fulkerson

Este algoritmo es un problema de árboles generadores mínimos, consiste en conocer el árbol generador inicial y de cada candidato conocer el camino más largo de uno de sus ciudades al hospital. De esta manera, consiguiendo minimizar esto conseguimos encontrar el mejor lugar para el hospital pues para cualquier otro tendrá un mayor camino más corto. Para ello, báscular uso del greedy generado de árboles generadores mínimos llamado Kruskal.

Por tanto, dado un grafo $g(v,u)$ y dado un vector de nodos el algoritmo sería el siguiente

Nodo PosHospital (grafo & g(v,u), Nodo *v)

list<Nodos> arbolGeneradorMinimal = Kruskal (g(v,u)); //obtenemos el arbol generador minimal

Nodo bestCity = null; int min =

for (int i=0; i<v.size(); i++)

int maxPathToHospital = arbolGeneradorMinimal.getmaxPathToHospital(v[i]);

if (maxPathToHospital < min)

min = maxPathToHospital

bestCity = v[i];

}

{

return bestCity;

{

Este código se ha realizado como pseudocódigo pero replicaría la creación de la clase Nodo, la clase grafo si uno se representara como lista de nodos y la implementación de varios métodos