



## Grado en Informática Algorítmica

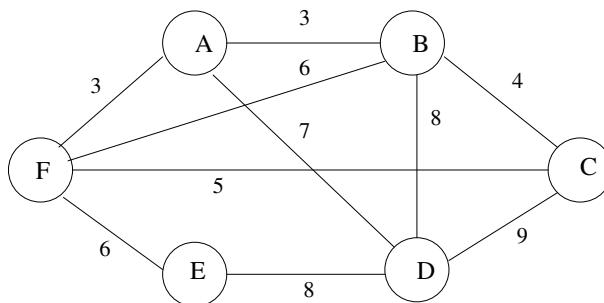
Curso 2018/2019. Convocatoria ordinaria de junio  
6 de junio de 2019

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

$$T(n) = 3T\left(\frac{n}{2}\right) + n \log_2(n)$$

con  $T(1) = 1$ .

2. (2 puntos) Una estación de ITV consta de  $m$  líneas de inspección de vehículos iguales. Hay un total de  $n$  vehículos que necesitan inspección. En función de sus características, cada vehículo tardará en ser inspeccionado un tiempo  $t_i$ ,  $i = 1, \dots, n$ . Se desea encontrar la manera de atender a los  $n$  vehículos y acabar en el menor tiempo posible. Diseñad un algoritmo de exploración de grafos que determine cómo asignar los vehículos a las líneas. Especificad claramente, además del algoritmo, la representación del problema, la representación de la solución, las restricciones explícitas e implícitas, así como las posibles cotas a utilizar.
3. (2 puntos) La empresa X ha decidido instalar una red de fibra óptica para interconectar sus seis centros de trabajo. El coste (positivo) del tendido de cable entre ellos aparece en la figura siguiente. Especificad un algoritmo que sea capaz de encontrar la forma de interconectar todos los centros con un coste mínimo, y aplicadlo para resolver el problema de la figura, describiendo el proceso paso a paso.



4. (2 puntos) Tenemos que pagar la cuenta en un restaurante por valor de  $M$  euros, y disponemos de una cantidad ilimitada de monedas de  $n$  tipos diferentes, siendo  $c[i]$  el valor de cada moneda de tipo  $i$ ,  $i = 1, \dots, n$ . Tenemos prisa y no queremos esperar que, en su caso, nos den la vuelta, pero como no hemos quedado muy satisfechos queremos pagar o la cantidad exacta  $M$  o la mínima cantidad posible mayor que  $M$  (o sea dejar ninguna propina o la mínima propina posible de acuerdo a las monedas que tenemos). Diseñar un algoritmo eficiente que determine la cantidad mínima que tenemos que pagar y cómo hacerlo.

Aplicadlo para resolver el siguiente caso: hay  $n = 3$  tipos de monedas de valores 5, 7 y 13, y queremos pagar una cantidad de  $M = 11$  unidades.

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

$$T(n) = 3T\left(\frac{n}{2}\right) + n \log_2(n)$$

con  $T(1) = 1$ .

Vemos a aplicar un cambio de variable claro:

$$u = 2^m \text{ con } m \in \mathbb{N} \Leftrightarrow \log_2 u = m$$

De manera que nuestra ecuación pasa a ser:

$$T(2^m) = 3T(2^{m-1}) + 2^m \cdot \log_2(2^m) = 3T(2^{m-1}) + m2^m$$

Solucionamos la parte homogénea

$$T(2^m) = 3T(2^{m-1}) \Leftrightarrow t_m - 3t_{m-1} = 0, p(\lambda) = (\lambda - 3) \text{ luego } R_0 = 3, w_0 = 1$$

De la función de ajuste obtenemos:

$$w_0 2^m = p(m) \cdot b^m, p(m) = m, b = 2 \text{ luego } R_1 = 2, w_1 = 2$$

Por tanto la ecuación se reduce a:

$$\begin{aligned} R_0 &= 3, w_0 = 1 \\ R_1 &= 2, w_1 = 2 \end{aligned} \quad \Rightarrow \quad (x-3)(x-2)^2 = 0$$

$$\text{Por tanto } T(m) = c_0 3^m + c_1 2^m + c_2 m 2^m$$

Deshaciendo el cambio de variable:

$$T(u) = c_0 3^{\log_2 u} + c_1 2^{\log_2 u} + c_2 \log_2 u 2^{\log_2 u} = c_0 u^{\log_2 3} + c_1 u + c_2 u^2$$

luego  $T(u) \in O(u^2)$

2. (2 puntos) Una estación de ITV consta de  $m$  líneas de inspección de vehículos iguales. Hay un total de  $n$  vehículos que necesitan inspección. En función de sus características, cada vehículo tardará en ser inspeccionado un tiempo  $t_i$ ,  $i = 1, \dots, n$ . Se desea encontrar la manera de atender a los  $n$  vehículos y acabar en el menor tiempo posible. Diseñad un algoritmo de exploración de grafos que determine cómo asignar los vehículos a las líneas. Especificad claramente, además del algoritmo, la representación del problema, la representación de la solución, las restricciones explícitas e implícitas, así como las posibles cotas a utilizar.

Tenemos un problema de minimización y usaremos Branch & Bound. Disponemos de la siguiente información

Suponemos  $n = m = 4$

$W \setminus L$	0	1	2	3
0	$t_{00}$	$t_{01}$	$t_{02}$	$t_{03}$
1	$t_{10}$	$t_{11}$	$t_{12}$	$t_{13}$
2	$t_{20}$	$t_{21}$	$t_{22}$	$t_{23}$
3	$t_{30}$	$t_{31}$	$t_{32}$	$t_{33}$

Autos de hablar de los costos, hablaremos de la representación de las soluciones:

Sea  $S$  una solución al problema, entonces si tiene 4 componentes, será un vector donde cada posición representa un vehículo y el contenido de la componente, será la llave de inspección a la que ir.

Además, supondremos que el problema puede no tener solución, si no hay llaves de ITU suficientes.

### Hablando de autos

• Global: Será la suma de los tiempos ocasionados por el algoritmo greedy aplicado al problema. Este algoritmo se encarga de elegir en cada paso, el menor tiempo del conjunto de candidatos. No proporciona la solución óptima.

• Local: Se usará el tiempo ya planificado y como estimación, se sumará el menor tiempo de cada uno de los vehículos, una sola solución factible; pero como mínimo será ese tiempo.

### Algoritmo:

BreadthFirst (vector tiempos)

Cola con Prioridad de nodos q; nodo raíz; vector sol; sol insertar raiz

q.insertar(raiz)

while (!q.empty()) {

nodo = q.pop();

q.insertar(nodo.hijos);

} // [sol[sol.size() - 1] != 1 && es solución]}

Actualiza Superior () //actualiza la cota superior con la solución  
return sol;

else {

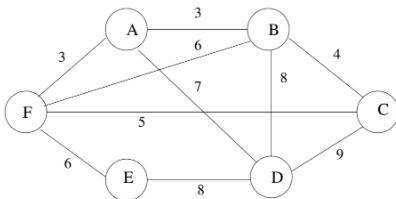
if (cota\_inferior(sol, q.top, tiempos) < cota\_global) {

//Procesamos el nodo mas prometedor

else q.pop() // lo quitamos sin procesar los hijos

```
{  
}  
{  
    return vectorado;  
}
```

3. (2 puntos) La empresa X ha decidido instalar una red de fibra óptica para interconectar sus seis centros de trabajo. El coste (positivo) del tendido de cable entre ellos aparece en la figura siguiente. Especificad un algoritmo que sea capaz de encontrar la forma de interconectar todos los centros con un coste mínimo, y aplicadlo para resolver el problema de la figura, describiendo el proceso paso a paso.



Este problema es una clara aplicación del algoritmo de Kruskal. Este algoritmo es un algoritmo greedy basado en la búsqueda de la unión de todos los nodos de un grafo conexo obteniendo el camino mínimo.

- **Tendidos: aristas**
- **Seleccionados: aristas ya elegidas que forman componentes conexas**
- **Función de factibilidad:** El problema tendrá una solución factible cuando el grafo sea conexo
- **Función de selección:** Tomaremos la arista de menor costo de manera que no se produzcan ciclos.
- **Función objetivo:** Consiguir que todos los nodos estén conectados, sin formar ciclos.

Veremos aprobar que nuestro algoritmo da la solución óptima.

- **Primer elección:** Sea  $S_0$  una solución óptima y aleatoria por si a la arista elegida en la iésima decisión. Veremos suponer que  $S_0$  es la mejor elección. Como el nodo origen es fijo para un caso particular entonces  $E(S_0)$  es la arista de menor costo como primera elección y queda otra solución. Por tanto, si  $S'_0$  es esa arista, obtendremos que el costo de  $(S_0 \cup S'_0)$  es menor que el costo de  $S_0$ , llegando a que

$S_u$  es óptima

→ Estructuras óptimas. Si  $S$  es solución óptima del grafo con vértices  $\{v_0, \dots, v_n\}$ .

Vemos que  $S \setminus \{v_u\}$  es solución óptima al problema  $\{v_0, \dots, v_{u-1}, v_{u+1}, \dots, v_n\}$ . Supongamos que no lo es, entonces  $\exists S'$  solución al problema  $\{v_0, \dots, v_{u-1}, v_{u+1}, \dots, v_n\}$  con costo menor. Por tanto, si incluimos el vértice  $v_u$  obtendremos:

$$\text{costo}(S) = \text{costo}(S \setminus \{v_u\}) + \text{costo}(v_u)$$

$$\text{costo}(S' \cup v_u) = \text{costo}(S') + \text{costo}(v_u)$$

Luego  $\text{costo}(S' \cup v_u) \leq \text{costo}(S)$ , es decir,  $S_u$  es solución óptima.

El algoritmo es el siguiente:

```
→ struct Arista;
    → bool hayCiclo (list<Aristas> s); // comprueba si la lista de aristas
        → Nodo u1, u2;
        → double peso;
    { → bool esConexo (list<Aristas> s), // comprueba que el grafo
        → s es conexo
    }
```

```
list<Aristas> visual (list<Aristas> grafo, list<Aristas> sol, int uuu - uodos)
```

```
int uodosconectados [uuu - uodos];
```

```
if (!sol.empty()) sol.clear();
```

```
while (!grafo.empty()) // asume que grafo es conexo
```

```
Arista bc = grafo.seleccionaBestCandidate(), // selecciona la arista de menor tamaño;
```

```
sol.push_back(bc);
```

```
if (hayCiclo(sol))
```

```
sol.pop_back();
```

```
{
```

```
else
```

```
ActualizaNodos (sol, uodosconectados); // pone a 1 la posición del vértice que se ha conectado
```

```
if (uodosconectados (sol) == uuu - uodos && esConexo (sol))
```

```
return sol;
```

```
{
```

```
{
```

```
return sol;
```

```
{
```

4. (2 puntos) Tenemos que pagar la cuenta en un restaurante por valor de  $M$  euros, y disponemos de una cantidad ilimitada de monedas de  $n$  tipos diferentes, siendo  $c[i]$  el valor de cada moneda de tipo  $i$ ,  $i = 1, \dots, n$ . Tenemos prisa y no queremos esperar que, en su caso, nos den la vuelta, pero como no hemos quedado muy satisfechos queremos pagar la cantidad exacta  $M$  o la mínima cantidad posible mayor que  $M$  (o sea dejar ninguna propina o la mínima propina posible de acuerdo a las monedas que tenemos). Diseñar un algoritmo eficiente que determine la cantidad mínima que tenemos que pagar y cómo hacerlo.

Aplicando para resolver el siguiente caso: hay  $n = 3$  tipos de monedas de valores 5, 7 y 13, y queremos pagar una cantidad de  $M = 11$  unidades.

$5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$

↓DN

Este problema se puede solucionar con Backtracking o B&B. Puedes hacerlo usando una vez B&B, usando Backtracking.

- Global: Usaremos un algoritmo greedy que da el subóptimo
- Inferior: Dado una secuencia usaremos como cota inferior, el valor ya gastando siempre tantas veces como sea posible dentro del valor restante el valor de la moneda más pequeña. Recorremos que es una muy mala cota inferior

La solución se representaría por un vector de  $n$  componentes, donde cada componente representa el número de monedas de ese valor, devolverá un vector. Si en el sistema monetario,  $v[0]$  es el uno céntimos de 1 cént. y  $v[5]$  el número de monedas de 50 cént.

Algoritmo:

```

void PagaExacto (vector<int> valores, int cuenta, int *sol) {
    if (sol.valor() >= cuenta) sol.Suprimir(); // si la suma de los componentes ya es solución borramos arriba
    else {
        // sol debe ser un vector de n componentes inicializados a 0 o cualquier otro valor positivo
        u=0; // incluye u en
        while (!uTodosBuenosProbados (sol,u)) // este bucle suma 1 a una posición seguidamente que referencia
            // la posición y resta 1 al anterior generando el próximo hijo
        if (sol.Factible())
            PagaExacto (valores, cuenta, sol);
        }
        k++;
    }
}

```



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 - 18071 - Granada (España)

5. (2 puntos) Un aficionado al póker juega una partida cada día, y anota los euros que gana o pierde. Por ejemplo, la siguiente tabla muestra los resultados del último mes.

L	M	X	J	V	S	D
					29	-7
14	21	30	-47	1	7	-39
23	-20	-36	-41	27	-34	7
48	35	-46	-16	32	18	5
-33	27	28	-22	1	-20	-42

DyV → El mes que es la práctica.  
→ ↗ ↘

Podemos ver que empezó el mes con una ganancia de 29 euros, y terminó con una pérdida de 42. El beneficio total obtenido en el mes (la suma de todas las ganancias y pérdidas) es -50 euros. Analizando la información, el jugador se podría dar cuenta de que si hubiera empezado a jugar el día 16 y terminado el día 26, habría maximizado sus ganancias, obteniendo 105 euros.

En general, dado un vector de ganancias/pérdidas de longitud  $n$ , se desea encontrar el subvector sobre el cual se consigue el beneficio total máximo. Diseñad un algoritmo que realice esta tarea de forma lo más eficiente posible.

**Duración del examen:** 2 horas y 30 minutos.

Práctica 2 cambio de soluciones → DyV posterior por la unidad