UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Algorítmica (11593)

**Segundo parcial**

20/enero/2023

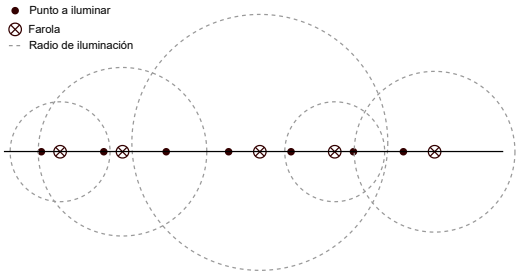
Escuela Técnica Superior de Ingeniería Informática

etsinf

**NOMBRE:** _____ **NÚM.:** _____

**1** **2 puntos**

Hay $M$ farolas en las posiciones $f_1 \leq f_2 \leq \ldots \leq f_M$ de una recta y $N$ puntos $x_1 \leq x_2 \leq \ldots \leq x_N$, también en la recta, que hay que iluminar. Cada farola tiene un radio de iluminación $r_i$, tal que la $i$-ésima farola ilumina los puntos en el intervalo $[f_i - r_i, f_i + r_i]$. Se quiere encender el mínimo número de farolas tales que cada uno de los $N$ puntos $x_1, x_2, \ldots, x_N$ esté iluminado por al menos una farola. Encuentra este mínimo número. (Asumimos que al menos existe una solución.) El siguiente esquema ilustra una instancia del problema:




En el ejemplo de la figura hay 7 puntos a iluminar y 5 farolas. Una posible solución factible sería: usar las farolas [1, 3, 4, 5], pero buscamos la solución óptima, la que minimiza el número de farolas. **Se pide:**

a) Explica la estrategia voraz a seguir.
b) Diseña una función Python que utilice esa estrategia voraz para minimizar el número de faroles que iluminen todos los puntos. La función recibe una lista con la localización de los puntos, una lista con la localización de las farolas y una lista con el radio de cada farola. Debe devolver una lista con las farolas seleccionadas.
c) Calcula el coste temporal de tu algoritmo.
d) ¿Qué devuelve tu algoritmo para el ejemplo presentado?

**Solución:** Nótese que este algoritmo sigue el proceso de "elegir la mejor opción en cada momento": esta es la idea principal en la que se basan los algoritmos voraces, y en este caso, de forma análoga al problema del repostaje visto en clase, llega a una solución óptima.

Concretamente, para este problema, la estrategia voraz de elegir para cada punto $x_i$ la farola que más alcance tenga por la derecha, nos proporciona una solución óptima al problema. El algoritmo devuelve `[2,3,5]`, esto es, las farolas 2, 3 y 5.

```
# Puntos y farolas ya ordenados
def farolas(x,f,r):
    resul = []
    derecha = x[0]-1 # punto iluminado mas a la derecha
    ultima = -1 # indice ultima farola utilizada
    for pto in x: # recorremos puntos
        if pto > derecha: # primer punto sin iluminar
            # buscar farola que ilumine más a derecha pero ilumine punto
            derecha,ultima = max((f[i]+r[i],i)
                                 for i in range(ultima+1,len(f))
                                 if f[i]-r[i] <= pto)
            resul.append(ultima)
    return resul # devolvemos indices de farolas a encender

x = [3, 4, 4, 7, 10]
f = [1,2,3,3,7,8]
r = [1,4,3,2,1,2]
print(farolas(x,f,r))

>>>[2, 5]
```

Esta solución itera para todas las farolas en $O(M)$ y entre las que lo cubren elegimos la que $f_i + r_i$ sea máximo. (En casos normales procesaremos bastante menos que $N$ puntos, como se puede ver en la figura). Sin embargo, en el peor de los casos procesamos $N$, y de aquí la complejidad $O(NM)$.

**2** **4 puntos**

Debemos reubicar $N$ camiones que se van a utilizar para escoltar distintos transportes de mercancía por carretera que salen de $N$ campamentos. Cada camión está en un campamento diferente y, por seguridad, se debe trasladar a cualquier otro campamento, excepto el campamento origen, y al final, en cada campamento debe haber un único camión. Por ejemplo, si hay 4 campamentos, el camión ubicado en el campamento 1 puede ser reubicado en cualquier otro campamento que no sea el 1. Los mandos nos proporcionan una matriz con el coste $C$ de reubicar el camión de cada posible campamento origen a cada posible campamento destino. Un ejemplo de matriz de costes $C$ sería:

| $orig \backslash dest$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | - | 1 | 4 | 2 |
| 2 | 3 | - | 4 | 2 |
| 3 | 5 | 5 | - | 3 |
| 4 | 7 | 8 | 5 | - |

**Se pide:** diseña un algoritmo de Ramificación y poda que calcule la reubicación de todos los camiones que suponga un menor coste global.

a) Expresa el problema en términos de optimización: conjunto de soluciones factibles, la función objetivo a minimizar y la solución óptima. Explica brevemente cómo expresas una solución $x$ del conjunto $X$. Escribe un ejemplo de solución para la instancia dada.

b) Describe los siguientes conceptos sobre los estados que serán necesarios para el algoritmo:
   1) Representación de un estado (no terminal). Escribe un ejemplo de estado para la instancia dada.
   2) Condición (o condiciones) para que un estado sea solución. Escribe un ejemplo de estado solución para la instancia dada.
   3) Identifica el estado inicial que representa todo el conjunto de soluciones factibles. Escribe cómo sería para la instancia dada.

c) Define una función de ramificación. Contesta a las siguientes cuestiones:
   1) Explica la función.
   2) Define la función (en python o en lenguaje matemático).
   3) Escribe un ejemplo de ramificación de un estado para la instancia dada.

d) Diseña una cota optimista no trivial. Contesta a las siguientes cuestiones:
   1) Explica la cota (caso general, de un estado intermedio). Calcula la cota para un estado ejemplo de la instancia dada.
   2) Explica el cálculo de la cota del estado inicial. Calcula la cota del estado inicial para la instancia dada.
   3) Define la cota (en python o en lenguaje matemático). Calcula el coste temporal.
   4) Estudia si se puede mejorar el cálculo de la cota, haciéndolo incremental. Define la cota así definida (en python o en lenguaje matemático). Calcula el coste temporal. Calcula la cota de forma incremental para un estado ejemplo de la instancia dada.

**Solución:** El conjunto de soluciones factibles o espacio de búsqueda serían todas las posibles asignaciones de camiones a campanentos, siempre que un camión de un campamento $i$ no se quede en el campamento $i$: esto es, todas las permutaciones de $N$ elementos, $(x_1, x_2, \ldots, x_N)$, eliminando aquellas donde $x_i = i$. La función objetivo será la suma de los costes: $f((x_1, x_2, \ldots, x_N)) = \sum_{i=1}^{N} C[i, x_i]$, y buscamos la solución mínima. Un ejemplo de solución para la instancia dada sería: $(2, 3, 4, 1)$.

Un estado intermedio se puede representar como una permutación incompleta: $(x_1, x_2, \ldots, x_k, ?) = (2, 3, ?)$, con $k < N$. El estado inicial será la tupla vacía $(?)$, pues el primer camión puede reubicarse en cualquier campanento menos en el de origen. Cuando $k = N$, la permutación será completa (con las restricciones impuestas) y llegamos a una solución. Definido así el estado, la función *branch* para un estado ramificable será:

$$branch((x_1, x_2, \ldots, x_{k-1}, ?)) = \{(x_1, x_2, \ldots, x_{k-1}, x_k, ?) \mid 1 \leq x_k \leq N; x_k \neq x_i, 1 \leq i < k; x_k \neq k\}$$

Para el estado inicial, la ramificación devolverá $N-1$ estados, pues el primer camión puede reubicarse en cualquier campanento menos el primero:

$$branch((?)) = \{(x_1, ?) \mid 1 < x_1 \leq N\}$$

Un ejemplo para la instancia presentada será:

$$branch((?)) = \{(2, ?), (3, ?), (4, ?)\}$$

$$branch((2, 3, ?)) = \{(2, 3, 1, ?), (2, 3, 4, ?)\}$$

Una posible cota optimista para este problema se puede obtener sumando al coste de la parte conocida, el menor coste posible para cada camión que queda por reubicar, independientemente de que el destino ya esté ocupado:

$$F((x_1, x_2, \ldots, x_k, ?)) = \sum_{1 \leq i \leq k} C[i, x_i] + \sum_{k+1 \leq i \leq N} \min_{1 \leq j \leq N, j \neq i} C[i, x_j]$$

Esta cota inferior se puede obtener en tiempo constante si:

1) hacemos un preproceso que calcula el coste mínimo para cada camión, lo que supone un coste $O(N^2)$, y lo almacenamos en un vector: $minC[i] = \min_{1 \leq j \leq N, j \neq i} C[i, x_j]$. Para la instancia dada: $minC = [1, 2, 3, 5]$.
2) se calcula de forma incremental de la forma siguiente:

$$F((x_1, x_2, \ldots, x_{k-1}, x_k, ?)) = F((x_1, x_2, \ldots, x_{k-1}, ?)) + C[k, x_k] - minC[k]$$

El cálculo de la cota en el estado inicial consistirá en sumar el coste mínimo para cada camión, lo que supone un coste $O(N)$, pero a partir de ahí, el cálculo de la cota supone un coste constante. Para la instancia presentada, $F((?)) = 1+2+3+5 = 11$. Y para un estado intermedio, $F(3, ?) = F(?)+C[1, 3]-minC[1] = 11 + 4 - 1 = 14$.

**3** **4 puntos**

Tenemos $N$ trabajos que pueden ser realizados por la máquina $M1$ o por la $M2$, si bien cada máquina tarda un tiempo distinto en realizar cada trabajo. Por ejemplo, si $N = 4$, una asignación de trabajos se representa mediante una tupla de $N$ componentes $(x_1, x_2, \ldots, x_N) = (2, 2, 1, 2)$, esto es, el trabajo $tr3$ se asigna a la máquina $M1$, y el resto de trabajos a la máquina $M2$. El tiempo total de una asignación viene dado por el máximo tiempo que necesita cada una de las máquinas para procesar todos sus trabajos. Por ejemplo, la asignación $(2, 2, 1, 2)$ con la siguiente tabla de tiempos

| máquina \ trabajo | $tr1$ | $tr2$ | $tr3$ | $tr4$ |
|---|---|---|---|---|
| $M1$ | 4 | 4 | 3 | 5 |
| $M2$ | 2 | 3 | 4 | 4 |

necesita: para la máquina $M1$, el tiempo requerido para el trabajo $tr3$ es $t_{1,3} = 3$; y para la máquina $M2$, los tiempos de los trabajos $tr1$, $tr2$ y $tr4$, esto es, $2 + 3 + 4 = 9$; con lo que el tiempo total será el máximo de ambos, 9. Se desea obtener la asignación de trabajos a máquinas que minimice el tiempo total.

Haz una traza de una algoritmo de Ramificación y Poda para la instancia presentada que use como cota optimista (cota inferior, en este caso) de un estado $(x_1, x_2, \ldots, x_k, ?)$ la siguiente función:

$$F(x_1, x_2, \ldots, x_k, ?) = \frac{1}{2}\left(\sum_{i=1}^{k} t_{i,x_i} + \sum_{i=k+1}^{N} \text{mín}(t_{i,1}, t_{i,2})\right)$$

La idea que subyace es: como el trabajo $i$-ésimo consumirá un tiempo que será, al menos, $\text{mín}(t_{i,1}, t_{i,2})$ es imposible que se realicen todos los trabajos en un tiempo menor que si se reparten estos tiempos entre ambas máquinas para acabar simultáneamente.

Para la traza, ten en cuenta lo siguiente:

- Sigue una estrategia por primero el mejor (en caso de empate, el estado más cercano a una solución) y usa poda explícita o implícita. Indica cuál usas.
- La traza debe mostrar el *conjunto de estados activos* de cada iteración del algoritmo indicando la cota optimista de cada estado (ej: como superíndice) y subrayando o marcando el estado que se selecciona para la siguiente iteración. Hay que indicar también si se actualiza la variable mejor solución y la poda implícita u otras podas en cada iteración que se produzca.
- Si para la traza utilizas el esquema que inicializa la variable mejor solución $\hat{x}$ a una solución factible, describe qué algoritmo o método utilizas para calcularla y cuál es su coste temporal.

**Solución:** Un ejemplo de solución factible ya se ha dado en el enunciado: $(2, 2, 1, 2)$. En realidad, el conjunto de soluciones factibles son las tupla de tamaño $N$ con cualquier combinación de 1's y 2's): $(1, 2, 1, 2)$, $(2, 1, 1, 2)$, $(1, 1, 1, 1)$, ...

Los estados intermedios los representaremos con tuplas incompletas, $(1, ?)$, $(2, 2, ?)$, ..., y la ramificación siempre obtiene dos nuevos estados: asignar el siguiente trabajo a $M1$ y a $M2$. Por ejemplo, $branch((2, 2, ?)) = \{(2, 2, 1, ?), (2, 2, 2?)\}$

La función objetivo de una solución $(x_1, x_2, \ldots, x_N)$ será:

$$f((x_1, x_2, \ldots, x_N)) = \text{máx}\left(\sum_{i:x_i=1} t_{i,1}, \sum_{i:x_i=2} t_{i,2}\right)$$

Podemos hacer un preproceso y precalcular el tiempo menor para cada trabajo, independiente de la máquina, y quedaría: $minT = [2, 3, 3, 4]$. Inicializamos el conjunto de estados activos con la tupla vacía, y su cota será: $1/2(2 + 3 + 3 + 4) = 6$. Comencemos:

- $A^0 = \{\overbrace{(?)}^{6}\}$

- La cota se puede calcular de forma incremental:

$$F((?)) = 1/2(2+3+3+4) = 6$$

$$F((1,?)) = 1/2((4)+(3+3+4)) = F((?)) + 1/2(t_{11} - minT[1]) = 6 + 1/2(4-2) = 7$$
$$F((2,?)) = 1/2((2)+(3+3+4)) = F((?)) + 1/2(t_{12} - minT[1]) = 6 + 1/2(2-2) = 6$$

  De forma general: $F((x_1,\ldots,x_k,?)) = F((x_1,\ldots,x_{k-1},?)) + 1/2(t_{kx_k} - minT[k])))$

  $A^1 = \{ \overbrace{(1,?)}^{6+1/2(4-2)=7} , \underline{\overbrace{(2,?)}^{6+1/2(2-2)=6}} \}$

- $A^2 = \{ \overbrace{(2,1,?)}^{6+1/2(4-3)=6.5} , \underline{\overbrace{(2,2,?)}^{6+1/2(4-4)=6}}, \overbrace{(1,?)}^{7}\}$

- $A^3 = \{ \underline{\overbrace{(2,2,1,?)}^{6-1/2(3-3)=6}}, \overbrace{(2,2,2,?)}^{6-1/2(4-3)=6.5} , \overbrace{(2,1,?)}^{6.5}, \overbrace{(1,?)}^{7}\}$

- Al ramificar el estado $(2,2,1,?)$ se obtienen dos soluciones factibles: $(2,2,1,1)$ (de valor 8) y $(2,2,1,2)$, de valor 9. Guardamos $(2,2,1,1)$.

  $A^4 = \{\underline{\overbrace{(2,2,2,?)}^{6.5}}, \overbrace{(2,1,?)}^{6.5}, \overbrace{(1,?)}^{7}\}$

- Al ramificar el estado $(2,2,2,?)$ se obtiene: $(2,2,2,1)$, de valor 11, y $(2,2,2,2)$, de valor 13. Ambas se descartan.

  $A^5 = \{\overbrace{(1,?)}^{7}, \underline{\overbrace{(2,1,?)}^{6.5}}\}$

- $A^6 = \{ \underline{\overbrace{(2,1,1,?)}^{6.5+1/2(3-3)=6.5}}, \overbrace{(2,1,2,?)}^{6.5+1/2(4-3)=7} , \overbrace{(1,?)}^{7}\}$

- Obtenemos dos nuevas soluciones: $(2,1,1,1)$, de valor 12, que se descarta, y $(2,1,1,2)$, de valor 7, que guardamos.

  En la siguiente iteración se elige $(2,1,2,?)$ para explorar y como no puede mejorar la mejor solución, el algoritmo termina.