

Análisis de algoritmos
16 de febrero de 2010 (semipresencial)

Prueba individual

Nombre:

Grupo:

Análisis de algoritmos

Dada una matriz M de tamaño $m \times n$ y un valor num , los dos siguientes algoritmos buscan algún valor de los que contiene la matriz que aparezca exactamente num veces en una fila o en una columna:

```
1 def busca_valor_1(M, num):
2     m=len(M)
3     n=len(M[0])
4     for fil in xrange(m): #Para cada fila de la matriz...
5         for col in xrange(n):
6             candidato=M[fil][col] #obtiene los posibles candidatos y comprueba si alguno aparece num veces.
7             cuenta=0
8             for col2 in xrange(n):
9                 if M[fil][col2]==candidato:
10                    cuenta+=1
11            if cuenta==num:
12                return candidato
13     for col in xrange(n): #Para cada columna de la matriz...
14         for fil in xrange(m):
15             candidato=M[fil][col] #obtiene los posibles candidatos y comprueba si alguno aparece num veces.
16             cuenta=0
17             for fil2 in xrange(m):
18                 if M[fil2][col]==candidato:
19                     cuenta+=1
20             if cuenta==num:
21                 return candidato
22     return None
```

```
1 def busca_valor_2(M, num):
2     m=len(M)
3     n=len(M[0])
4     for fil in xrange(m):
5         for col in xrange(n):
6             candidato=M[fil][col] #Para cada posible candidato de la matriz...
7             for fil2 in xrange(m): #comprueba si en alguna fila aparece exactamente num veces, y
8                 cuenta=0
9                 for col2 in xrange(n):
10                    if M[fil2][col2]==candidato:
11                        cuenta+=1
12                if cuenta==num:
13                    return candidato
14            for col2 in xrange(n): #comprueba si en alguna columna aparece exactamente num veces.
15                cuenta=0
16                for fil2 in xrange(m):
17                    if M[fil2][col2]==candidato:
18                        cuenta+=1
19                if cuenta==num:
20                    return candidato
21     return None
```

Indica y justifica cuál es el coste temporal y espacial asintótico de ambos, distinguiendo entre el mejor y el peor de los casos si los hubiera (señala ante qué entradas se comportarían en el mejor caso y cuando en el peor caso). Si tuvieras que elegir uno de los dos algoritmos para resolver el problema, ¿con cuál te quedarías? Justifícalo.

Estructuras de datos
23 de febrero de 2010 (semipresencial)

Prueba colectiva

Grupo:

Asistentes:

Selección de estructuras de datos eficientes para algoritmos

Una empresa ha puntuado a cada uno de sus N empleados en función del trabajo realizado durante el año y ha decidido otorgar una bonificación a los n mejores empleados ($n \ll N$). No obstante, para no discriminar a ninguna de las m secciones en que se divide su producción, ha decidido asignar un mínimo de p bonificaciones los p mejores empleados de cada una de las secciones (hay que tener en cuenta que $n > pm$).

Con esto, el procedimiento seguido consiste en bonificar a los p mejores trabajadores de cada sección y otorgar las restantes bonificaciones ($n - pm$) a los trabajadores de la empresa que presentan mejor puntuación, independientemente de la sección en la que estén.

Debes seleccionar las estructuras de datos adecuadas y describir con la mayor precisión un algoritmo que solucione el problema de la manera más eficiente posible, indicando y justificando el coste temporal asintótico asociado al mismo.

Divide y vencerás
2 de marzo de 2010 (semipresencial)

Prueba individual

Nombre:

Grupo:

1. Algoritmo mergesort

30 puntos

El siguiente algoritmo es una versión aleatorizada de la primera variante presentada de *mergesort*:

```
1 from random import randrange
2
3 def rdm_mergesort(a):
4     if len(a) <= 1:
5         return a
6     else:
7         j = randrange(1, len(a))
8         return merge(rdm_mergesort(a[:j]), rdm_mergesort(a[j:]))
```

(La llamada *randrange(1, len(a))* —línea 7— devuelve un número aleatorio entre 1 y $len(a) - 1$.)

Responde a las siguientes preguntas, *justificando tu respuesta en todos los casos*:

- a) ¿Cuál es el coste temporal del algoritmo en el mejor y en el peor de los casos (si los hay)?
- b) ¿Cuál es el coste espacial del algoritmo en el mejor y en el peor de los casos (si los hay)?
- c) ¿Se podría reducir alguno de los costes anteriormente citados si en lugar de generar dos subvectores se trabajara con índices sobre el vector? ¿Cuáles serían los costes mejorados resultantes?
- d) ¿Presenta el algoritmo recursión por cola? Si es así, elimínala e indica si ello afecta a alguno de los costes asintóticos previamente calculados y en que sentido.

2. Resolución de relaciones de recurrencia (Teorema maestro y corolarios)**30 puntos**

Encuentra (justificándola) una cota asintótica superior para cada una de las siguientes relaciones de recurrencia, donde $T(n)$ presenta un valor constante en $n = 1$ y cuyo término recursivo general (para $n > 1$) es:

a) $T(n) = 4T(n/2) + nc_1 + c_2.$

b) $T(n) = T(n/2) + \sqrt{nc_1}.$

c) $T(n) = 2T(n/3) + n^2c_1 + n \lg nc_2.$

d) $T(n) = 3T(n/3) + n \lg nc_1 + nc_2.$

(Nota: c_1 y c_2 son constantes positivas arbitrarias.)

Divide y vencerás y búsqueda con retroceso
16 de marzo de 2010 (semipresencial)

Prueba colectiva

Grupo:
Asistentes:

Divide y vencerás: problema

90 puntos

Sea a un vector que contiene valores de tres tipos: negativos, positivos y **None**. Queremos saber *si el vector contiene más valores negativos que positivos*. Para resolver el problema debéis tener en cuenta que todos los valores numéricos de un mismo tipo se encuentran agrupados en uno de los extremos del vector en zonas contiguas, mientras que todos los valores **None** están agrupados entre los dos grupos de valores numéricos. He aquí algunos ejemplos de vectores que cumplen esos requisitos:

a

-3	-6	-5	-2	-1	-5	None	None	None	6	4	9	1	7
----	----	----	----	----	----	------	------	------	---	---	---	---	---

a

3	4	None	-1	-3	-2	-6	-4	-2
---	---	------	----	----	----	----	----	----

a

-2	None	None	None	None	None	None	None	9	9
----	------	------	------	------	------	------	------	---	---

en los dos primeros casos la respuesta sería **True**, mientras que en el tercero sería **False**. Observad que los valores negativos y los positivos pueden estar indistintamente tanto al principio como al final del vector.

- a) Diseñad un algoritmo *empleando la técnica recursiva de divide y vencerás* que reciba un vector a con las características previamente descritas e indique si contiene más valores negativos que positivos. El algoritmo desarrollado debe ser *lo más eficiente posible*, tanto desde el punto de vista de la complejidad temporal como de la espacial.

Fijaos que lo que se pide es que la *estrategia fundamental* con la que resolvais el problema sea la de divide y vencerás, no que haya un único algoritmo de divide y vencerás. Dependiendo del enfoque elegido, puede que os lleve a diseñar una o varias funciones (de divide y vencerás) que sean empleadas para poder devolver el valor solicitado. En cualquier caso, no olvidéis incluir todo el código necesario para la resolución completa del problema.

- b) El algoritmo que habéis desarrollado ¿tiene mejor y peor caso o se comporta de manera uniforme? Describid con claridad el mejor y el peor caso si los hubiera y realizad los correspondientes análisis de complejidad temporal y espacial justificando los resultados obtenidos.
- c) Indicad si el algoritmo desarrollado (o alguna de sus funciones) presenta recursividad por cola y si su transformación en un algoritmo iterativo daría lugar a la variación de alguno de los costes calculados en el anterior apartado.

Si os sobra tiempo, podríais intentar resolver también la siguiente **cuestión adicional**:

- d) En el caso en que vuestro algoritmo (o alguna de sus funciones) presente recursividad por cola, eliminadla. (Si empleais más de una función que presenta recursividad por cola, basta con que la eliminéis en una de ellas.)

Divide y vencerás y búsqueda con retroceso
16 de marzo de 2010 (semipresencial)

Prueba individual

Nombre:

Grupo:

Búsqueda con retroceso: traza del problema de las n reinas

50 puntos

En el problema de las n reinas hay que disponer n reinas en un tablero de $n \times n$ escaques de modo que ninguna amenace a otra. El método `backtracking` de la clase `NQueensSolver1` (página 6-7) permite encontrar una solución factible para el problema aplicando la técnica de búsqueda con retroceso:

```
class NQueensSolver1:

    def is_complete(self, s):
        return len(s) == self.n

    def is_promising(self, s):
        return all(s[j] != s[i] and j-i != abs(s[j]-s[i]) for i in xrange(len(s)) \
                  for j in xrange(i+1, len(s)))

    def backtracking(self, s):
        if self.is_complete(s): return s
        for row in xrange(self.n):
            s' = s + [row]
            if self.is_promising(s'):
                found = self.backtracking(s')
                if found != None: return found
        return None
```

El método también permite encontrar una solución factible que tenga un prefijo determinado.

Realiza una traza de la ejecución del método para la llamada `NQueensSolver1(5).backtracking([3,1])` (es decir, el problema de ubicar 5 reinas sin que se amenacen en un tablero de 5×5 donde las dos primeras tienen que ocupar las posiciones 3 y 1 respectivamente).

Debes dibujar el árbol de estados incluyendo tan sólo aquellos *que se generan* durante el proceso de búsqueda. Incluye también los estados que se estudian pero se descartan por no superar la condición de ser prometedores, y márcalos de forma especial. Indica claramente cuál es el resultado devuelto por el algoritmo.

Búsqueda con retroceso
23 de marzo de 2010 (semipresencial)

Prueba colectiva

Grupo:

Asistentes:

Problema

100 puntos

Dado un grafo $G = (V, E)$, diseña un algoritmo de *búsqueda con retroceso* que encuentre un camino que pase por todas las aristas del grafo *una sola vez* (sin repetir ninguna arista). Para ello se pide que:

- a) Describas cómo vas a representar los estados en la resolución del problema planteado (en qué van a consistir las tuplas: cuántos elementos van a contener, valor posible de esos elementos).
- b) Escribas un algoritmo de búsqueda con retroceso que solucione el problema, indicando cuál sería la llamada inicial que harías para resolverlo.
- c) A la vista del código desarrollado en el apartado anterior, ¿cuál sería la complejidad espacial del algoritmo desarrollado? ¿Y la complejidad temporal en el *mejor de los casos*? Justificalas.

Si os sobra tiempo, podríais intentar también responder a las siguientes **cuestiones adicionales**:

1. ¿Es el algoritmo que has desarrollado el más eficiente posible? Si crees que puedes realizar algunas modificaciones que permitan reducir su coste, descríbelas (y justificalas).
2. Una variante del problema anterior consistiría en, dado un grafo $G = (V, E)$ y dos vértice u y v , encontrar el camino que pase por todas las aristas del grafo sin repetir aristas y que empiece en un vértice u y acabe en un vértice v . ¿Serías capaz de adaptar tu algoritmo para que resolviera dicho problema?

Algoritmos voraces
30 de marzo de 2010 (semipresencial)

Prueba individual

Nombre:

Grupo:

Desglose óptimo de una cantidad de dinero

20 puntos

Dados los sistemas monetarios con monedas de valores:

- a) 1, 2, 4, 16.
- b) 1, 2, 5, 10, 20, 25.
- c) 1, 3, 5.
- d) 1, 5, 10, 20.

Indica y justifica, para cada uno de ellos, si el algoritmo voraz `greedy_change` proporciona siempre (ante cualquier entrada válida) la solución óptima (si tu respuesta es negativa deberás encontrar un contraejemplo que lo demuestre).

Algoritmos voraces
30 de marzo de 2010 (semipresencial)

Prueba colectiva

Grupo:
Asistentes:

El problema de la mochila con fraccionamiento

25 puntos

El problema de la mochila con N objetos que se pueden fraccionar se resuelve eficientemente con una estrategia voraz con un coste temporal de $O(N \lg N)$. ¿Es posible efectuar alguna modificación en cada uno de estos supuestos para obtener algoritmos más eficientes? Si es así, indica cómo y acompaña la explicación con el coste de los algoritmos resultantes.

- a) Todos los productos pesan lo mismo.
- b) Todos los productos valen lo mismo.
- c) Todos los productos pesan y valen lo mismo.
- d) La mochila no tiene límite de capacidad.
- e) De cada producto se puede cargar una cantidad arbitraria en la mochila (es decir, se pueden cargar tanto fracciones de objetos como varias unidades de los mismos).

Programación dinámica
10 de mayo de 2010 (presencial)

Prueba individual

Nombre:

Grupo:

Problema

50 puntos

Deseamos colocar una valla de M metros en línea recta para marcar el linde entre dos terrenos. La construcción de la valla la haremos ensamblando, una tras otra, piezas de valla prefabricadas que podemos adquirir en promoción en una gran superficie. Hay piezas disponibles de N longitudes distintas. De cada tipo de pieza i ($1 \leq i \leq N$) conocemos su longitud, $l(i)$, su precio, $p(i)$, y el número de unidades que quedan en stock, $s(i)$. La promoción indica que al hacer una compra de piezas estamos obligados a adquirir un mínimo de una pieza de cada tipo, por lo que ya sabemos que al menos deberemos emplear una de cada en la construcción de la valla.

Deseamos conocer qué piezas hemos de comprar para construir una valla de longitud M con el menor coste posible. La siguiente ecuación recursiva devuelve el coste mínimo asociado a la mejor combinación de piezas:

$$C(i, m) = \begin{cases} 0, & \text{si } i = 0 \text{ y } m = 0; \\ +\infty, & \text{si } i = 0 \text{ y } m > 0; \\ +\infty, & \text{si } i > 0 \text{ y } m < l(i); \\ \min_{1 \leq x \leq \min(\lfloor m/l(i) \rfloor, s(i))} C(i-1, m-x \cdot l(i)) + x \cdot p(i), & \text{si } i > 0 \text{ y } m \geq l(i). \end{cases}$$

donde la llamada $C(N, M)$ proporciona la mínima inversión necesaria para construir una valla de M metros gastando al menos una pieza de cada uno de los N tipos disponibles. Se pide:

- a) Representa el grafo de dependencias asociado a la ecuación recursiva para un problema con $M = 9$ metros de valla y $N = 3$ tipos de piezas con las siguientes longitudes $l = [1, 2, 3]$, precios $p = [2, 3, 4]$ y unidades en stock $s = [4, 3, 1]$.
- b) Escribe un algoritmo iterativo asociado a la anterior ecuación que permita obtener el valor del coste más económico con el que podemos construir la valla.
- c) Indica cuál es el coste temporal y espacial del algoritmo previo justificando brevemente tu respuesta.

Programación dinámica
10 de mayo de 2010 (presencial)

Prueba colectiva

Grupo:
Asistentes:

Problema

80 puntos

Sea el anterior problema:

Deseamos colocar una valla de M metros en línea recta para marcar el linde entre dos terrenos. La construcción de la valla la haremos ensamblando, una tras otra, piezas de valla prefabricadas que podemos adquirir en promoción en una gran superficie. Hay piezas disponibles de N longitudes distintas. De cada tipo de pieza i ($1 \leq i \leq N$) conocemos su longitud, $l(i)$, su precio, $p(i)$, y el número de unidades que quedan en stock, $s(i)$. La promoción indica que al hacer una compra de piezas estamos obligados a adquirir un mínimo de una pieza de cada tipo, por lo que ya sabemos que al menos deberemos emplear una de cada en la construcción de la valla.

Deseamos conocer qué piezas hemos de comprar para construir una valla de longitud M con el menor coste posible. La siguiente ecuación recursiva devuelve el coste mínimo asociado a la mejor combinación de piezas:

$$C(i, m) = \begin{cases} 0, & \text{si } i = 0 \text{ y } m = 0; \\ +\infty, & \text{si } i = 0 \text{ y } m > 0; \\ +\infty, & \text{si } i > 0 \text{ y } m < l(i); \\ \min_{1 \leq x \leq \min(\lfloor m/l(i) \rfloor, s(i))} C(i-1, m-x \cdot l(i)) + x \cdot p(i), & \text{si } i > 0 \text{ y } m \geq l(i). \end{cases}$$

donde la llamada $C(N, M)$ proporciona la mínima inversión necesaria para construir una valla de M metros gastando al menos una pieza de cada uno de los N tipos disponibles.

Sobre el problema original se pide:

- a) Escribid el algoritmo iterativo que proporciona la secuencia de decisiones óptima, concretamente, debe proporcionar el número de piezas de cada tipo que debemos comprar para efectuar el mínimo gasto.

A continuación, y debido a la finalización de la promoción en la superficie comercial, se producen las siguientes modificaciones en el enunciado del problema:

1. No hay obligación de comprar al menos una pieza de cada tipo.
2. El stock de piezas de cada tipo es ilimitado (pueden pedir las que necesitemos a la fábrica).

El resto de condiciones se mantienen.

Para este nuevo problema se pide:

- a) Plantead la ecuación recursiva de programación dinámica que calcule, de la forma más eficiente posible, la mínima inversión que debemos realizar para construir la valla, indicando cuál debe ser la llamada inicial que se efectúe a dicha ecuación para obtener el valor óptimo.
- b) Representad el grafo de dependencias entre las llamadas de la ecuación recursiva para los datos anteriores: $M = 9$ metros de valla y $N = 3$ tipos de piezas con las siguientes longitudes $l = [1, 2, 3]$ y precios $p = [2, 3, 4]$.
- c) Indicad cuál sería el coste temporal y espacial de un algoritmo iterativo diseñado a partir de vuestra ecuación recursiva que devolviera la *secuencia de decisiones óptima*. Justificad brevemente la respuesta.
- d) ¿Sería posible reducir el coste espacial del apartado anterior si el algoritmo iterativo sólo tuviera que devolver el valor del máximo beneficio? Si es así, indicad y justificad cuál sería dicho coste.

Puntuación extra: Formalizad el nuevo problema en términos de optimización.

Prueba individual

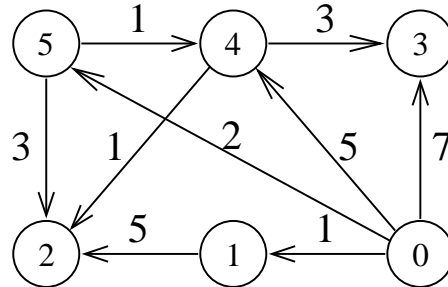
Nombre:

Grupo:

Algoritmo de Dijkstra: traza

35 puntos

Dado el siguiente grafo dirigido y ponderado:



- a) Haz una traza, iteración a iteración, de la ejecución de `all_shortest_paths_from_source2` (pág. 7-48, la primera versión del algoritmo de Dijkstra que utiliza punteros hacia atrás) sobre el grafo anterior para encontrar los caminos más cortos desde el vértice 0 a todos los demás. Para ello, rellena la siguiente tabla (haciendo uso de las filas que necesites):

iter	<i>D</i>						<i>backpointer</i>						<i>frontier</i>	<i>added</i>
	0	1	2	3	4	5	0	1	2	3	4	5		
inic.														
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
...														

- b) Indica cuál es el *árbol* de caminos más cortos desde el vértice 0 a todos los demás que devuelve el algoritmo (puedes dibujarlo sobre el grafo).

Prueba colectiva

Grupo:
Asistentes:

Adaptación del algoritmo de Huffman

30 puntos

El algoritmo de Huffman permite obtener una codificación binaria prefija para los n símbolos de un alfabeto de los que se conoce su frecuencia de aparición. Para ello construye un árbol binario. Dicho árbol cumple la propiedad de que, de entre todos los posibles, es aquel que minimiza el número esperado de bits en la codificación de los símbolos del alfabeto. El coste temporal del algoritmo es $O(n \lg n)$.

Estudia cada uno de los cuatro supuestos que se plantean a continuación. Para cada caso, indica si es posible modificar el algoritmo de Huffman o plantear alguna estrategia alternativa que iguale o mejore el coste de este. Si es así, indica cómo (señalando las estructuras de datos necesarias para la resolución) y acompaña la explicación con el coste temporal de los algoritmos resultantes:

- a) Los símbolos del alfabeto se nos proporcionan inicialmente ordenados de menor a mayor frecuencia de aparición.
- b) Los símbolos del alfabeto presentan todos la misma frecuencia de aparición.
- c) Los símbolos del alfabeto cumplen que hay uno de ellos que se presenta la mitad de las veces (su frecuencia es $1/2$), otro se presenta un cuarto de las veces ($1/4$), otro $1/8$, otro $1/16$ y así sucesivamente. Los símbolos se nos facilitan desordenados.
- d) El mismo caso que el anterior, pero los símbolos se nos proporcionan ordenados de mayor a menor frecuencia.

En el caso de los supuestos b) y c), ¿podrías justificar cuál es el número máximo de bits que se utilizarán para representar un símbolo? ¿y el número mínimo?

Algoritmos voraces y programación dinámica

20 de abril de 2010 (semipresencial)

Prueba colectiva

Grupo:

Asistentes:

Análisis de la solución voraz de un problema

65 puntos

Un profesor con m alumnos en su asignatura ha preparado nm trabajos. Cada alumno debe hacer n trabajos. A cada alumno se le ha pedido que puntue cada trabajo entre 0 y 10 según el interés que para él tendría realizar el trabajo. Por tanto, se dispone de una matriz P , donde cada celda $P[a][t]$ contiene la puntuación que el alumno a ($1 \leq a \leq m$) otorga al trabajo t ($1 \leq t \leq nm$).

Deseamos asignar trabajos a los alumnos de modo que la satisfacción global (suma de puntuaciones) sea máxima. Para resolver el problema hemos diseñado las siguientes *estrategias voraces*:

1. Se considera el primer alumno y se le asignan los n trabajos que prefiere; a continuación, se asignan al segundo alumno sus n trabajos preferidos de entre los no asignados al primero; y así sucesivamente.
2. Para cada alumno se calcula la suma de las puntuaciones de sus n trabajos preferidos y se ordena a los alumnos a partir de dicho valor de mayor a menor valor; considerando ese orden, se aplica el método del apartado anterior.
3. Se construye un máx-heap con todas la puntuaciones de la matriz (junto a las que se almacenan el alumno y trabajo asociados a cada puntuación); se van extrayendo los elementos del máx-heap y asignando los trabajos a los alumnos según el orden de extracción hasta que todos los alumnos tengan asignados sus n trabajos (si se extrae un trabajo que le corresponde a un alumno que ya tiene asignados sus n trabajos o se extrae un trabajo que ya ha sido asignado a otro alumno, dicho elemento se descarta).

Debes justificar *para cada estrategia* si es correcta (si está garantizado que ante una entrada válida *siempre* devuelve la solución óptima) o no (en el caso en que no lo sea, debes demostrarlo con un contraejemplo) y debes indicar, razonándolo, el coste temporal de la estrategia.

Algoritmos voraces y programación dinámica

20 de abril de 2010 (semipresencial)

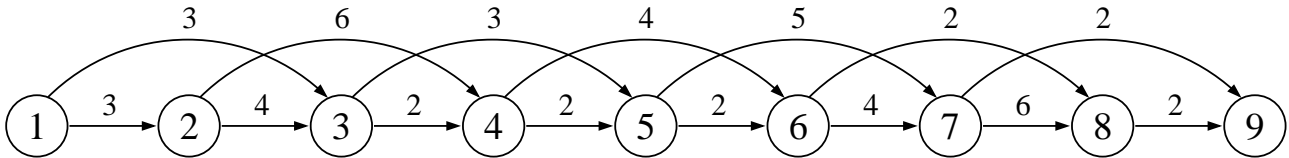
Prueba individual

Nombre:

Grupo:

Programación dinámica: traza del problema del trayecto más barato en el río Congo 50 puntos

Debes encontrar cuál es la secuencia de embarcaderos óptima entre el 1 y el 9 para la siguiente instancia del problema del trayecto más barato en el río Congo:



Para ello, se pide:

- Haz una traza del algoritmo `cheapest_price` (pág 7-10). Para ello, bien puedes utilizar como base el grafo anterior de forma similar a cómo se hace en el apartado (g) de la figura 7.8 (pág. 7-11), bien rellenar las estructuras C y B del algoritmo. No se pide una traza iteración a iteración, basta con que se muestre el contenido final de las estructuras citadas.
- ¿Cuál es el valor de la solución óptima? ¿En qué estructura de las anteriores y en qué posición dentro de esta se almacena? ¿Qué resultado devuelve el algoritmo? ¿Qué significa este último resultado?

Prueba individual

Nombre:

Grupo:

Problema de la mochila discreta

75 puntos

Dada una mochila con capacidad para cargar $W = 4$ unidades de peso y $N = 5$ objetos que podemos cargar en ella, con pesos $w = [2, 1, 3, 2, 1]$ y valores $v = [10, 4, 13, 15, 7]$, se pide:

- a) Dibuja, aprovechando la siguiente cuadrícula, el *grafo extendido de dependencias entre estados* asociado a dicha instancia del problema (utiliza las cajas que necesites, tachando el resto, y traza los arcos entre estados correspondientes al grafo para el problema propuesto):

(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)
(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)

- b) Haz una traza del algoritmo `knapsack` (pág 7-22), el que permite recuperar la secuencia óptima de objetos. Para ello, escribe arriba de cada uno de los estados del grafo del apartado anterior el valor que el algoritmo almacena en el diccionario V y debajo del estado el valor que se almacena en el diccionario de punteros hacia atrás para el estado (alternativamente, puedes utilizar flechas tal y como se hacía en la figura 7-16).
- c) ¿Cuál es el valor de la solución óptima? ¿En qué estado del grafo de dependencias se almacena? ¿Qué resultado devuelve el algoritmo? ¿Qué significa este último resultado?

Primer control parcial de Esquemas Algorítmicos (IG24)

27 de marzo de 2010

1. Divide y vencerás

5 puntos

Sea a un vector que contiene valores numéricos que pueden aparecer repetidos. Los elementos se distribuyen en orden ascendente hasta una determinada posición, a partir de la cual aparecen en orden descendente. Además, todas las repeticiones de un mismo elemento están *agrupadas* ocupando posiciones contiguas dentro del vector. Deseamos encontrar la posición del elemento que tiene mayor valor. He aquí algunos ejemplos de vector, en los que se destacan en negrita las posiciones dentro del vector que se corresponden con el elemento de mayor valor (en los casos en que haya más de una, basta con indicar una cualquiera):

	0	1	2	3	4	5	6	7	8	9				
a	1	3	3	3	11	11	6	6	6	4				
a	2	6	18	15	15	15	15	7	7	7	7	1	1	1
a	18	15	15	15	15	7	7	7	7	6	2	1	1	1
a	1	1	1	2	6	7	7	7	7	15	15	15	15	18

Fíjate que en algunos casos puede aparecer más de una vez el elemento máximo (primer ejemplo), mientras que en otros (los dos últimos) el valor máximo puede estar en uno de los extremos del vector.

Debes resolver el problema de, dado un vector a tal y como se ha descrito, *averiguar cuál es la posición que ocupa el elemento con mayor valor*.

- a) Diseña un algoritmo que reciba el vector a y resuelva el problema *mediante la técnica recursiva de divide y vencerás*. El algoritmo desarrollado debe ser *lo más eficiente posible*, tanto desde el punto de vista de la complejidad temporal como de la espacial.

Fíjate que lo que se pide es que la *estrategia fundamental* con la que resuelvas el problema sea la de divide y vencerás, no que haya un único algoritmo de divide y vencerás. Dependiendo del enfoque elegido, puede que te lleve a diseñar una o varias funciones (de divide y vencerás) que sean empleadas para poder devolver el valor solicitado. En cualquier caso, no olvides incluir todo el código necesario para la resolución completa del problema.

Pista: puede resultarte útil, dada la posición de un elemento en el vector, desarrollar y utilizar alguna función para determinar en que posición se encuentra la primera y/o la última aparición de dicho elemento en el vector.

- b) El algoritmo que has desarrollado ¿tiene mejor y peor caso o se comporta de manera uniforme? Describe con claridad el mejor y el peor caso si los hubiera y realiza los correspondientes análisis de complejidad temporal y espacial justificando los resultados obtenidos.
- c) Si el algoritmo desarrollado (o alguna de sus funciones) presenta recursividad por cola, elimínala e indica si alguno de los costes calculados en el anterior apartado varía. (Si has implementado más de una función que presenta recursividad por cola, basta con que la elimines en una de ellas, aunque deberás indicar qué otras funciones la tienen y señalar las posibles modificaciones de los costes asumiendo que hubieras transformado en iterativas dichas funciones.)
- d) ¿Funcionaría tu algoritmo en el caso en el que los elementos repetidos no tuviesen que estar agrupados (es decir, que pudiesen aparecer indistintamente en el tramo de ascenso y en el de descenso)? Justifica brevemente tu respuesta.

2. Búsqueda con retroceso

5 puntos

Disponemos de N objetos deformables, cada uno con un volumen v_1, v_2, \dots, v_N , y de C cajas iguales, cada una con una capacidad volumétrica V , y nos gustaría encontrar, si existe, la forma de combinar los objetos para poder meterlos todos en las C cajas sin exceder su capacidad.

Queremos diseñar un algoritmo de *búsqueda con retroceso* capaz de resolver el problema. Se pide que:

- a) Describas cómo vas a representar los estados en la resolución del problema planteado (en qué van a consistir las tuplas: cuantos elementos van a contener, valor posible de esos elementos).
- b) Escribas un algoritmo de búsqueda con retroceso que resuelva el problema, indicando cuál sería la llamada inicial que harías para resolver el problema.
- c) A la vista del código desarrollado en el apartado anterior, ¿cuál sería la complejidad espacial del algoritmo desarrollado? ¿Y la complejidad temporal en el *mejor de los casos*? Justifícalas.
- d) ¿Es el algoritmo que has desarrollado el más eficiente posible? Si crees que puedes realizar algunas modificaciones que permitan reducir su coste, descríbelas (y justifícalas).
- e) ¿Se te ocurre alguna manera de averiguar cuál sería el menor número de cajas necesarias para poder almacenar todos los objetos?

Segundo control parcial de Esquemas Algorítmicos (IG24)

15 de mayo de 2010

1. Algoritmos voraces

3.25 puntos

En una asociación cultural con n socios, cada socio tiene la obligación de proponer un tema sobre el que puede impartir una conferencia para los miembros de la asociación que quieran asistir. Con el fin de reducir al mínimo el gasto, en la asociación han acordado *organizar el menor número posible de las conferencias propuestas*, con la condición de que se garantice que todos los socios participen al menos en una de ellas (sea como conferenciante o como público). Para ello se dispone de n listas: una por cada socio con las personas (incluido el propio socio conferenciante) a las que les gustaría asistir a la conferencia impartida por él. He aquí un ejemplo para $n = 8$ socios:

$$S1 = [S1, S2, S7], \quad S2 = [S1, S2, S6, S8], \quad S3 = [S3, S4, S6, S8], \quad S4 = [S3, S4, S5, S7], \quad S5 = [S4, S5], \\ S6 = [S2, S3, S6, S7, S8], \quad S7 = [S1, S4, S6, S7], \quad S8 = [S2, S3, S6, S8].$$

Una posible solución sería organizar las conferencias propuestas por los socios $S1$, $S2$, $S3$ y $S4$, ya que con esta combinación todos los socios imparten o asisten a alguna conferencia, aunque no sería la mejor solución, puesto que con los datos del ejemplo se podría encontrar una combinación con menor número de conferencias.

Para resolver el problema, y considerando inicialmente que los n socios son candidatos a dar su conferencia, se proponen estas dos Estrategias Voraces:

- EV1: Seleccionar, de entre los candidatos, el socio cuya conferencia tiene un mayor número de solicitudes de asistencia. Esa conferencia se impartirá. Eliminar de los candidatos a los socios apuntados a su conferencia (entre los que se encuentra él mismo). Repetir el proceso hasta que no queden socios candidatos.
- EV2: Buscar el socio candidato cuya conferencia tenga *menos* peticiones. De los socios que quieren asistir a ella, seleccionamos a aquel cuya conferencia tenga *más* solicitudes de asistencia y que aun sea candidato (si el socio inicial fuera el único que es candidato, él es el seleccionado). La conferencia del socio seleccionado se impartirá. Eliminar de los candidatos a los socios apuntados a su conferencia (entre los que se encuentra él mismo). Repetir el proceso hasta que no queden socios candidatos.

Se pide:

- Como se trata de un problema de optimización, describe qué características debería tener una solución para ser factible, cuál sería la función objetivo y qué debe cumplir una solución factible para ser la óptima.
- Aplica cada estrategia propuesta al problema del ejemplo e indica el resultado que proporcionaría, detallando en cada iteración el socio seleccionado y los descartados.
- Para cada estrategia propuesta, justifica si encuentra siempre una solución factible al problema y, en caso de hacerlo, si es siempre la óptima (debes demostrar con un contraejemplo los casos en que no esté garantizada la obtención de una solución factible y/u óptima).
- Para cada estrategia propuesta, señala el coste temporal que supone aplicarla (indica para ello las estructuras de datos que utilizarías en la implementación de un algoritmo lo más eficiente posible que siga la estrategia, describiendo los detalles del mismo que den lugar al coste señalado).

2. Programación dinámica

3.75 puntos

Tenemos una caja en la que guardamos sellos de S valores distintos, donde $v(s)$ es el valor del sello del tipo s ($1 \leq s \leq S$) y $P(s)$ es la cantidad de sellos de ese tipo que tenemos guardados. Queremos enviar una carta cuyo valor de franqueo es F . Como hemos acumulado demasiados sellos nos gustaría emplear la mayor cantidad posible posible, gastando al menos $p(s)$ sellos de cada tipo ($1 \leq s \leq S$).

Deseamos saber, aplicando la estrategia de *programación dinámica*, cuántos sellos de cada tipo emplearemos para completar el franqueo exacto F de la carta de forma que el número total de sellos utilizados sea el máximo posible con las restricciones previamente indicadas.

1. Bajo las condiciones anteriormente expuestas:

- Formaliza el problema en términos de optimización.
- Plantea la ecuación recursiva asociada que calcula el máximo número de sellos necesarios para realizar el franqueo. Indica cuál debe ser la llamada que se efectue a dicha función para obtener el valor óptimo.
- Representa el grafo de dependencias entre llamadas de la ecuación recursiva para un franqueo de $F = 8$, si tenemos $S = 4$ tipos de sellos, con valores $v = [1, 2, 3, 5]$, número de sellos de cada tipo disponible $P = [4, 4, 5, 2]$ y debiendo emplear al menos $p = [2, 1, 1, 0]$. Indica y razona los costes espacial y temporal con los que un algoritmo iterativo podría efectuar los cálculos para resolver el problema (indica explícitamente, justificando tu respuesta, si es posible efectuar una reducción de la complejidad espacial si nos interesara obtener tan sólo el número de sellos máximo).

d) ¿Podría haber instancias del problema para las que no existiera una solución factible? Razona la respuesta y, si es así, indica las condiciones que se deberían cumplir para que pueda haber una solución factible.

2. Una variante del problema sería la siguiente: no tengo necesidad de gastar un mínimo número de sellos de cada valor, por contra me gustaría quedarme con al menos $p'(s)$ sellos de cada valor para futuros envíos.

¿Cómo modificarías la formulación del problema para que recogiera este nuevo planteamiento? ¿Cuál sería ahora la ecuación recursiva que resuelve el problema y la llamada que harías a la misma? ¿Con qué coste temporal y espacial se podría resolver?

3. Programación dinámica

3 puntos

Queremos obtener el máximo beneficio posible alquilando una sala para la realización de una serie de actividades. Tenemos la opción de realizar A posibles actividades (numeradas entre 1 y A). De cada actividad a ($1 \leq a \leq A$) conocemos la hora de comienzo $s(a)$, la de terminación $t(a)$ y el número de participantes $p(a)$. En la sala no puede realizarse más de una actividad al mismo tiempo. Todas las actividades pagan un precio fijo por cada participante en la misma, por lo que el máximo beneficio lo obtendremos con la combinación de actividades que mayor número total de asistentes atraigan a la sala. El número total de horas disponible de la sala es H (por simplicidad, todos los datos relacionados con las horas de comienzo y terminación de las actividades, así como de las horas en que está disponible la sala se han normalizado a valores enteros entre 0 y H .)

Debes averiguar, empleando la técnica de programación dinámica, qué actividades debemos seleccionar para obtener el mayor número de asistentes a la sala. Como ayuda, la siguiente ecuación recursiva de programación dinámica permite resolver el problema de averiguar el número de asistentes máximo que podrá ocupar la sala:

$$P(h) = \begin{cases} 0, & \text{si } h = 0; \\ \max(P(h-1), \max_{\substack{1 \leq a \leq A: \\ t(a)=h}} P(s(a)) + p(a)), & \text{si } h > 0. \end{cases}$$

La llamada $P(H)$ proporciona dicho valor.

Se pide:

- Representa el grafo de dependencias para el problema en el que el número de actividades es $A = 6$, la hora de cierre de la sala es $H = 9$ y las horas de comienzo, terminación y número de participantes en cada actividad son $s.t.p = [(0,3,10), (2,4,8), (1,5,20), (5,7,9), (3,8,12), (4,8,15)]$ (cada triplete de la estructura $s.t.p$ es una actividad, el primer componente es la hora de comienzo s , el segundo la hora de finalización t y el tercero el número de participantes p).
- Diseña un algoritmo iterativo lo más eficiente posible que devuelva *las actividades* que se deben realizar para obtener el mayor número de participantes.
- Indica cuál es el coste temporal y espacial del algoritmo diseñado justificando brevemente tu respuesta. ¿Crees que sería posible reducir el coste espacial si únicamente nos interesará obtener el valor del número de participantes máximo? Razona tu respuesta y, si es positiva, indica cuál sería dicho coste y justifícalo.

Examen convocatoria ordinaria de Esquemas Algorítmicos (IG24)

11 de junio de 2010

Marca con una X (una única opción) la parte o partes de la asignatura a la que te presentas (sólo debes rellenarlo si has firmado el contrato del método de evaluación alternativo):

Parte 1 (Preg. 1 y 2) Parte 2 (Preg. 3 y 4) Partes 1 y 2 Renuncio a la evaluación alternativa

(Si lo dejas en blanco o no lo rellenas claramente se asumirá que renuncias a la evaluación alternativa y te presentas a la convocatoria ordinaria.)

1. Divide y vencerás

2,5 puntos

Dados dos vectores que contienen cada uno n valores enteros positivos no repetidos, a ordenado de menor a mayor y b ordenado de mayor a menor, decimos que sus elementos están “hermanados crecientemente” si al elemento i -ésimo de a le asociamos el elemento $(n-1)-i$ de b , para $0 \leq i < n$. A los elementos $a[i]$ y $b[n-1-i]$ les llamamos “hermanos”. Aquí tienes un ejemplo de dos vectores hermanados crecientemente con $n = 18$ elementos:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	2	4	8	10	12	15	23	32	40	42	44	58	60	72	84	89	93	102
b	113	100	97	96	85	73	71	62	54	53	42	30	22	14	8	7	5	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

para el que, por ejemplo, el hermano del 32 ($a[7]$) es el 42 ($b[(18-1)-7] = b[10]$) y el hermano del 102 ($a[17]$) es el 113 ($b[(18-1)-17] = b[0]$).

Debes diseñar un algoritmo de *divide y vencerás* que, dados dos vectores a y b hermanados crecientemente y un valor v , devuelva *el par de elementos hermanos de cada vector cuya suma sea más cercana a v* . Para los anteriores vectores y el valor $v = 155$, el algoritmo debería devolver (72, 85) cuya suma es 157, la más cercana a 155 entre hermanos y que se corresponde con los hermanos que ocupan las posiciones 13 en a y 4 en b (fíjate que $b[4]$ es el hermano de $a[13]$ porque $(18-1)-13=4$). Se pide:

- Diseña un algoritmo recursivo *lo más eficiente posible* mediante la técnica de divide y vencerás que permita resolver este problema.
- Analiza y justifica la complejidad temporal y espacial del algoritmo desarrollado en el apartado anterior, indicando si presenta un comportamiento uniforme o existe un mejor y un peor caso.

2. Búsqueda con retroceso

2,5 puntos

En una organización mafiosa las rencillas internas han puesto en peligro tanto la integridad física de sus M miembros como la supervivencia de la organización. Tras una reunión, deciden dividir el territorio que controlan en N zonas independientes, a las que asignarán a los distintos miembros de la organización (un mafioso sólo podrá estar asignado a una zona). Queremos saber qué mafiosos se instalan en cada zona, teniendo en cuenta que dos mafiosos no pueden estar en la misma zona si previamente han tenido alguna disputa.

Para resolver el problema puedes asumir que los mafiosos aparecen numerados del 1 al M , las zonas de la 1 a la N y que dispones de una estructura de datos consistente en un vector `mafiosos_odiados` en el que en cada posición, `mafiosos_odiados[i]` (para $1 \leq i \leq M$), se almacena un conjunto con los mafiosos con los que tiene cuentas pendientes el mafioso i y, por tanto, con los que no puede estar en la misma zona.

Debes desarrollar un algoritmo de *búsqueda con retroceso* que, dados M , N y la estructura `mafiosos_odiados`, devuelva el reparto de los mafiosos entre las zonas sin que estén en peligro (ellos y la organización). Se pide:

- Describe cómo vas a representar los estados en la resolución del problema planteado (en qué van a consistir las tuplas: cuántos elementos van a contener, valor posible de esos elementos).
- Escribe un algoritmo de búsqueda con retroceso que resuelva el problema, indicando cuál sería la llamada inicial que realizarías.
- A la vista del código desarrollado en el apartado anterior, ¿cuál sería la complejidad espacial del algoritmo desarrollado? ¿Y la complejidad temporal en el *mejor de los casos*? Justifícalas.

3. Algoritmos voraces

1,5 puntos

La programación diaria de una cadena de televisión que compite por tener una buena audiencia está compuesta por P programas (numerados entre el 1 y el P según el orden de emisión).

La cadena está obligada a clasificar los programas en F franjas o bloques de emisión consecutivos sabiendo que no es posible alterar el orden de emisión prefijado de los programas y que no puede haber más de M programas por franja. Es decir, a la primera franja irán los programas desde el 1 hasta el i (donde i es un valor variable, $1 \leq i \leq M$), en la segunda franja los programas desde el $i+1$ hasta el j ($i+1 \leq j \leq i+M$) y así sucesivamente.

Se dispone de estimaciones de la audiencia prevista para cada programa, a_1, a_2, \dots, a_P , la audiencia de una franja es la audiencia del programa con mayor audiencia incluido en la franja y la audiencia global de la cadena se calcula sumando las audiencias de todas las franjas (de sus programas más vistos).

Teniendo en cuenta todo lo anterior, se desea encontrar la distribución de los P programas entre las F franjas de emisión que maximiza la audiencia de la cadena. Para ello, se propone la siguiente *estrategia voraz*, que se puede aplicar en tres etapas consecutivas:

1. Averiguamos cuáles son los F programas con mayor audiencia y los almacenamos por orden de emisión en un vector (de tipo offset) **mejores**.
2. Asignamos los programas a las franjas horarias de la siguiente manera: los programas comprendidos entre el 1 y **mejores[1]** van a la primera franja, los que están entre **mejores[1]+1** y **mejores[2]** van a la segunda franja, y así sucesivamente excepto para la última franja, en la que metemos los programas que están entre **mejores[F-1]+1** y el programa P .
3. Comprobamos que no haya ninguna franja horaria que contenga más de M programas, si fuera así desplazaríamos los programas por exceso a la(s) franja(s) anterior(es) y/o posterior(es) hasta que todas las franjas contengan como máximo M programas.

Se pide:

- a) Ya que se trata de un problema de optimización, debes indicar las características que posee una *solución factible*, cuál es la función objetivo que se pretende optimizar y qué debe cumplir una solución factible para ser considerada óptima.
- b) Detalla cómo implementarías cada una de las etapas de la estrategia anterior para que el algoritmo resultante sea lo más eficiente posible: para cada etapa debes describir las características más importantes de su implementación y citar las estructuras de datos más significativas que utilizarías, así como el coste temporal de la misma. Finalmente, señala el coste temporal global del algoritmo.
- c) Justifica si la estrategia propuesta encuentra siempre una solución factible al problema y, en caso de hacerlo, si es siempre la óptima (debes demostrar con un contraejemplo el caso en que no esté garantizada la obtención de la solución factible y/u óptima).

4. Programación dinámica

3,5 puntos

Sea el problema presentado en el apartado de algoritmos voraces de encontrar la distribución de P programas de televisión entre F franjas de emisión que maximiza la audiencia de una cadena de televisión, con *todas las restricciones y condiciones allí fijadas* y con un par de modificaciones adicionales:

- Como mínimo hay que ubicar m programas en cada franja (además de no superar el máximo M).
- La audiencia de una franja no es la del programa con mayor audiencia de la franja, sino la media entre el programa de mayor audiencia y el de menor audiencia de la franja.

Deseamos diseñar un algoritmo mediante la estrategia de *programación dinámica* que determine qué programas ubicamos en cada franja para obtener la máxima audiencia. Se pide:

- a) Formaliza el problema en términos de optimización.
- b) Diseña una ecuación recursiva que calcule el valor de la máxima audiencia asociada a cualquier distribución de los programas entre las franjas, indicando cuál debe ser la llamada que se haga a dicha ecuación para resolver el problema.
- c) Diseña un algoritmo iterativo que resuelva el problema, devolviendo el valor de la máxima audiencia empleando la menor ocupación espacial (en términos asintóticos) posible. Indica y justifica sus costes espacial y temporal.

Examen segunda convocatoria de Esquemas Algorítmicos (IG24)

17 de septiembre de 2010

Marca con una X (una única opción) la parte o partes de la asignatura a la que te presentas (sólo debes rellenarlo si has firmado el contrato del método de evaluación alternativo):

Parte 1 (Preg. 1 y 2) Parte 2 (Preg. 3 y 4) Partes 1 y 2 Renuncio a la evaluación alternativa

(Si lo dejas en blanco o no lo rellenas claramente se asumirá que renuncias a la evaluación alternativa y te presentas a la convocatoria ordinaria.)

1. Divide y vencerás

2,5 puntos

Dado un vector a que contiene una secuencia con un número indeterminado de valores None en sus primeras posiciones, otra secuencia de None's en sus últimas posiciones y una serie de valores enteros positivos ordenados crecientemente en las posiciones que hay entre las dos secuencias de None's, queremos averiguar *las posiciones de a en que comienza y acaba la secuencia numérica*. He aquí algunos ejemplos de vectores como el descrito, en los que hemos oscurecido los valores de las posiciones buscadas:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	
a	None	None	None	1	2	4	7	12	None	None	None	None	None	None

0	1	2	3	4	5	6	7	8	
a	None	None	1	3	4	5	14	22	None

0	1	2	3	4	5	6	7	8	9	
a	None	None	None	None	None	None	None	13	29	None

Debes diseñar un algoritmo de *divide y vencerás* que, dado un vector a que cumple las condiciones señaladas, resuelva el problema de encontrar las posiciones inicial y final de la secuencia numérica. Se pide:

- Diseña un algoritmo recursivo *lo más eficiente posible* mediante la técnica de divide y vencerás que de solución a este problema.
- Analiza y justifica la complejidad temporal y espacial del algoritmo desarrollado en el apartado anterior, indicando si presenta un comportamiento uniforme o existe un mejor y un peor caso.

2. Búsqueda con retroceso

2,5 puntos

Disponemos de N objetos deformables, cada uno con un volumen v_1, v_2, \dots, v_N , y de C cajas, cada una con una capacidad volumétrica V_1, V_2, \dots, V_C , y nos gustaría encontrar, si existe, la forma de combinar los objetos para poder meterlos todos en las C cajas sin exceder su capacidad y siempre y cuando el número de objetos contenidos en cada caja no sea mayor de 10.

Queremos diseñar un algoritmo de *búsqueda con retroceso* capaz de resolver el problema. Se pide:

- Describe cómo vas a representar los estados en la resolución del problema planteado (en qué van a consistir las tuplas: cuantos elementos van a contener, valor posible de esos elementos).
- Escribe un algoritmo de búsqueda con retroceso que resuelva el problema, indicando cuál sería la llamada inicial que harías para resolver el problema.
- A la vista del código desarrollado en el apartado anterior, ¿cuál sería la complejidad espacial del algoritmo desarrollado? ¿Y la complejidad temporal en el *mejor de los casos*? Justifícalas.
- ¿Es el algoritmo que has desarrollado el más eficiente posible? Si crees que puedes realizar algunas modificaciones que permitan reducir su coste, descríbelas (y justifícalas).

3. Voraces

1,5 puntos

Dos equipos de n miembros, a y b , van a enfrentarse en un concurso de inteligencia. En el concurso, cada miembro de un equipo participa una única vez en un enfrentamiento cara a cara directo con un miembro del otro equipo. El ganador del enfrentamiento consigue un punto para su equipo. El objetivo de cada equipo es obtener la máxima cantidad de puntos.

Con antelación al comienzo del concurso, cada equipo dispone de los cocientes intelectuales de todos sus miembros (pero no de los cocientes de los miembros del otro equipo). Sin embargo, el mecanismo del concurso puede favorecer al equipo b , ya que antes de cada enfrentamiento individual el equipo a debe indicar qué miembro de su equipo va participar y su cociente y, una vez sabido esto, el equipo b elige al miembro de su equipo que se enfrentará al del equipo a . Asumiendo que no hay dos concursantes con el mismo cociente y que cuando se enfrentan dos personas lo normal es que gane la de mayor cociente, diseña una *estrategia voraz* que permita al equipo b ganar la mayor cantidad posible de puntos.

Se pide:

- Describe con la mayor precisión una estrategia voraz lo más eficiente posible que proporcione la máxima cantidad de puntos al equipo b . La estrategia deberá ser especialmente eficiente en el momento en que el equipo a facilite los datos de un concursante y el equipo b tenga que determinar qué miembro de su equipo concursará.
- Indica cuál es el coste temporal de la estrategia que has diseñado, justificándolo adecuadamente. Señala las estructuras de datos empleadas.

4. Programación dinámica

3,5 puntos

En un juego en el que nos enfrentamos varios amigos, en un instante dado dispongo de una baraja compuesta por C cartas, en la que cada carta c , $1 \leq c \leq C$, tiene puntos de ataque $p(c)$ y de defensa $p'(c)$. Para contrarrestar la jugada de un contrincante tengo que responder con la combinación de cartas que emplee el menor número de ellas y que proporcione una puntuación exacta de ataque de valor A y una de defensa de valor D .

Se pide:

- a) Formaliza el problema en términos de optimización,
- b) plantea la ecuación recursiva de *programación dinámica* que calcula el menor número de cartas necesario para alcanzar las puntuaciones indicadas, señalando cuál debe ser la llamada recursiva que se efectue a dicha ecuación para resolver el problema,
- c) diseña un algoritmo recursivo con memorización que devuelva el valor del número mínimo de cartas que necesito emplear, indicando y justificando sus costes espacial y temporal.