Análisis de algoritmos y Estructuras de datos
25 de febrero de 2009 (Mañana)

Prueba individual

Nombre: **Grupo:**

Análisis de algoritmos y estructuras de datos

La siguiente función, dados dos vectores de números sin repetir, a y b, devuelve una lista con los numeros comunes a los dos vectores siempre que haya k o más elementos iguales; en caso contrario devuelve dos listas con los elementos distintos de cada vector:

```
1  def iguales_o_distintos(a,b,k):
2      iguales=[]
3      distintos_a=[]
4      for x in a:
5          if x not in b:
6              distintos_a.append(x)
7          else:
8              iguales.append(x)
9      if len(iguales)>=k:
10         return iguales
11     else:
12         distintos_b=[]
13         for y in b:
14             if y not in a:
15                 distintos_b.append(y)
16     return (distintos_a,distintos_b)
```

a) Indica y justifica cuál es el coste temporal y espacial asintótico del algoritmo, distinguiendo entre el mejor y el peor de los casos si los hubiera (señala ante qué entradas se comportaría en el mejor caso y cuando en el peor caso).

b) ¿Podrías utilizar alguna(s) estructura(s) de datos auxiliar(es) que te permitiera(n) reducir el coste temporal del algoritmo propuesto? Si es así, indica cuál(es), cómo la(s) emplearías y el coste temporal resultante.

Prueba colectiva

Grupo:
Asistentes:

Estructuras de datos y costes

Nos plantean un problema en el que vamos a almacenar valores numéricos en una cola de prioridad implementada mediante un heap, pero debes modificarlo para que, *sin afectar al coste de las operaciones básicas sobre heaps* (inserción de un elemento, obtención del mínimo y extracción de este), sea posible, *con el menor coste temporal posible*:

a) conocer en cualquier momento el valor de la suma de todos los elementos del heap, y

b) que no se inserte en la cola un valor que en ese momento ya exista en el heap.

Para cada caso debes describir brevemente (y con claridad) cómo incorporarías la modificación efectuada, detallando cómo afecta a las operaciones básicas sobre heaps, asi como el coste temporal que implica realizar la operación asociada a la modificación introducida y si afecta al coste espacial del heap.

Análisis de algoritmos y Estructuras de datos
25 de febrero de 2009 (Tarde)

Prueba individual

Nombre: **Grupo:**

Análisis de algoritmos y estructuras de datos

La siguiente función comprueba si en un vector de números a existen al menos k elementos mayores o iguales que uno dado x. Si es así, devuelve los k elementos del vector mayores y más cercanos a x:

```
1  def k_mayores_mas_cercanos(a,x,k):
2      b=[a[i] for i in xrange(len(a)) if a[i]>=x]
3      if len(b)<k:
4          return None
5      elif len(b)==k:
6          return b
7      else:
8          c=[None]*k
9          for i in xrange(k):
10             c[i]=min(b)
11             b.remove(min(b))
12         return c
```

a) Indica y justifica cuál es el coste temporal y espacial asintótico del algoritmo, distinguiendo entre el mejor y el peor de los casos si los hubiera (señala ante qué entradas se comportaría en el mejor caso y cuando en el peor caso).

b) ¿Podrías utilizar alguna estructura de datos auxiliar que te permitiera reducir el coste temporal del algoritmo propuesto? Si es así, indica cuál, cómo la emplearías y el coste temporal resultante.

Prueba colectiva

Grupo:
Asistentes:

Estructuras de datos y costes

Nos plantean un problema en el que tenemos necesidad de utilizar una estructura de datos para trabajar con multiconjuntos (conjuntos en los que un mismo elemento puede aparecer más de una vez). Concretamente, necesitaremos utilizar las siguientes operaciones, que queremos que se ejecuten de la forma *más eficiente posible*:

a) añadir un elemento a la estructura,

b) preguntar si un elemento está en la estructura,

c) eliminar un elemento de la estructura (si hubiera más de una aparición de dicho elemento, eliminar una cualquiera de ellas), y

d) eliminar todas las apariciones de un elemento en la estructura.

Debes señalar brevemente (y con claridad) cómo implementarías esta estructura (puedes tomar como base otra u otras estructuras que conozcas), detallando cómo diseñarías cada operación y cuál sería el coste temporal de cada una de ellas. Señala también el coste espacial asociado a la estructura.

Divide y vencerás
9 de marzo de 2009 (Mañana)

Prueba individual

Nombre: **Grupo:**

1. Algoritmo *min_max* (traza de la recursión)

Dado el vector:

	0	1	2	3	4	5	6	7	8
a	3	8	5	9	4	2	10	7	8

dibuja el árbol de recursión que generaría la llamada al algoritmo de divide y vencerás *min_max*(*a*) (página 5-28). Para cada llamada debes señalar tanto los valores de los parámetros i y k en la función recursiva, como el valor devuelto por dicha llamada. ¿Cúal es el valor finalmente devuelto por el algoritmo?

2. Planteamiento y análisis de un problema

Dado un vector a que contiene valores enteros diferentes ordenados de menor a mayor, queremos averiguar cuantos de dichos valores cumplen que $i = a[i]$. Un ejemplo de vector sería el siguiente:

	0	1	2	3	**4**	**5**	**6**	7	8
a	-3	-2	0	2	**4**	**5**	**6**	9	11

en el que hay 3 valores (el 4, el 5 y el 6) que cumplen el criterio indicado (fijate en que, tanto en este caso como en cualquier otro, si hay varios valores que cumplen esa condición necesariamente deben aparecer en posiciones consecutivas del vector).

Describe con la mayor claridad posible cómo aplicarías la estrategia de divide y vencerás para abordar y resolver el problema de la forma más eficiente posible. Indica asimismo cuál crees que sería el coste temporal asociado a la estrategía planteada, dando una breve justificación del mismo.

Divide y vencerás
9 de marzo de 2009 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema

Dado un vector a que contiene valores enteros diferentes ordenados de menor a mayor, queremos averiguar cuantos de dichos valores cumplen que $i = a[i]$. Un ejemplo de vector sería el siguiente:

	0	1	2	3	**4**	**5**	**6**	7	8
a	-3	-2	0	2	**4**	**5**	**6**	9	11

en el que hay 3 valores (el 4, el 5 y el 6) que cumplen el criterio indicado (fijate en que, tanto en este caso como en cualquier otro, si hay varios valores que cumplen esa condición necesariamente deben aparecer en posiciones consecutivas del vector).

a) Diseñad un algoritmo recursivo *mediante la técnica de divide y vencerás* que devuelva el número total de valores que cumplen la condición descrita. El algoritmo desarrollado debe ser lo más eficiente posible, tanto desde el punto de vista de la complejidad temporal como de la espacial.

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) Modificad vuestro algoritmo para que reciba un valor u que representa un umbral. Cuando se efectue una llamada al algoritmo en la que el tamaño del vector que se vaya a analizar sea menor o igual que dicho umbral, el algoritmo deberá sustituir las llamadas recursivas por una búsqueda secuencial sobre el correspondiente fragmento de vector en la que se resuelva directamente el problema que se solucionaba recursivamente.

Divide y vencerás
9 de marzo de 2009 (Tarde)

Prueba individual

Nombre: **Grupo:**

1. Algoritmo *partition* (traza)

Haz la traza del comportamiento del algoritmo de partición de un vector empleado en *quicksort* (pág. 5-36) para la llamada: *partition*`([13, 4, 14, 9, 8, 12, 9, 14, 22, 16, 15, 18, 25, 22, 17], 0, 7)`.

Sobre la siguiente tabla debes detallar el valor de las variables i y j y el estado del vector justo antes de entrar en el bucle, después de cada una de las iteraciones del bucle y al final de la ejecución del algoritmo (si lo prefieres, puedes rellenar únicamente los valores de las casillas que se modifican entre una iteración y la siguiente). Marca también de forma especial la posición del pivote antes y después de la ejecución del algoritmo:

		vector														
j	i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		13	4	14	9	8	12	9	14	22	16	15	18	25	22	17

2. Planteamiento y análisis de un problema

Dados dos vectores con n componentes, a, con valores ordenados de menor a mayor, y b, con valores ordenados de mayor a menor, queremos averiguar, de todas las i posiciones del vector ($0 \leq i < n$), cuál es la posición en la que los valores de $a[i]$ y $b[i]$ están más cercanos. En el siguiente ejemplo,

	0	1	2	3	**4**	5	6	7	8
a	-3	-2	0	2	**4**	8	10	12	15
b	13	10	7	6	**5**	3	1	-2	-4

la posición 4 del vector es en la que los valores están más cerca.

Describe con la mayor claridad posible cómo aplicarías la estrategia de divide y vencerás para abordar y resolver el problema de la forma más eficiente posible. Indica asimismo cuál crees que sería el coste temporal asociado a la estrategía planteada, dando una breve justificación del mismo.

Divide y vencerás
9 de marzo de 2009 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema

Dados dos vectores con n componentes, a, con valores ordenados de menor a mayor, y b, con valores ordenados de mayor a menor, queremos averiguar, de todas las i posiciones del vector ($0 \leq i < n$), cuál es la posición en la que los valores de $a[i]$ y $b[i]$ están más cercanos. En el siguiente ejemplo,

	0	1	2	3	**4**	5	6	7	8
a	-3	-2	0	2	**4**	8	10	12	15
b	13	10	7	6	**5**	3	1	-2	-4

la posición 4 del vector es en la que los valores están más cerca.

a) Diseñad un algoritmo recursivo *mediante la técnica de divide y vencerás* que devuelva el valor i en el que $a[i]$ y $b[i]$ son más cercanos. El algoritmo desarrollado debe ser lo más eficiente posible, tanto desde el punto de vista de la complejidad temporal como de la espacial.

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) ¿Presenta vuestro algoritmo recursión por cola? Si es así, eliminadla e indicad si ello afecta a alguno de los costes asintóticos previamente calculados y en que sentido.

Búsqueda con retroceso
25 de mayo de 2009 (Mañana)

Prueba individual

Nombre: **Grupo:**

Problema de las n reinas: variante **50 puntos**

En la variante del problema de las n reinas conocida como el problema de las n reinas *cojas*, la definición del problema es similar a la del de las n reinas, excepto en que las reinas (cojas) también amenazan a cualquier otra reina que esté en la misma fila o columna, pero no a las de las diagonales; en lugar de eso, amenazan a cualquiera que se ubique en una casilla vecina a ella (cada casilla tiene un máximo de 8 casillas vecinas).

Indica cómo modificarías el algoritmo que resuelve el problema de las n reinas (no cojas) para que resuelva el de las n reinas cojas. Puedes partir de cualquiera de los algoritmos (o de las funciones que estos utilizan) que hemos visto en clase siempre que señales qué versión estás usando e indiques claramente y con detalle los cambios que realizas en ella.

Búsqueda con retroceso
25 de mayo de 2009 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

El juego del dominó dispone de 28 piezas rectangulares de igual tamaño divididas en dos partes. En cada parte de la pieza aparece un número de puntos (desde 0 puntos hasta 6 puntos). No hay dos piezas con la misma combinación de puntos en sus dos partes, y entre todas engloban toda posible combinación de puntos (desde el (0,0) hasta el (6,6)).

Podemos juntar dos o más piezas de dominó uniendo dos de sus partes libres por un extremo sólo si el valor de las partes unidas es el mismo (la pieza (0,1) no se podría juntar con la (2,3), pero sí con la (2,1), de esta manera, (0,1)(1,2); a partir de ahí, podríamos ahora unir, por ejemplo, la pieza (2,4) o la (0,4) a las dos anteriores, pero no la (1,4)). Quisieramos, partiendo de una pieza cualquiera, juntar todas las piezas del juego de forma correcta.

Para ello debes diseñar un algoritmo de *búsqueda con retroceso* que devuelva como resultado una secuencia de piezas que cumpla las condiciones comentadas.

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 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*? Justifícalas.

Si os sobra tiempo, podríais intentar responder también 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 justifícalas).

2. Una variante del problema anterior consistiría en que la última pieza de la secuencia cumpla que pueda unirse a la primera. ¿Serías capaz de adaptar tu algoritmo para que resolviera dicho problema?

Búsqueda con retroceso
25 de mayo de 2009 (Tarde)

Prueba individual

Nombre: **Grupo:**

Problema de la suma del subconjunto: variante **50 puntos**

En una variante del problema de la suma del subconjunto, además del peso se considera también el volumen: dados N objetos con pesos $w_1, w_2, \ldots, w_N$ y volumen $v_1, v_2, \ldots, v_N$ y una mochila con capacidad para soportar una carga W y un volumen V, deseamos cargar una selección arbitraria de objetos cuyo peso sea exactamente W sin sobrepasar el volumen máximo admitido por la mochila.

Indica cómo modificarías el algoritmo que resuelve el problema de la suma del subconjunto para que resuelva esta nueva versión. Puedes partir de cualquiera de los algoritmos (o de las funciones que estos utilizan) que hemos visto en clase siempre que señales qué versión estás usando e indiques claramente y con detalle los cambios que realizas en ella.

Búsqueda con retroceso
25 de mayo de 2009 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

En cada una de las seis caras de los dados empleados en los juegos infantiles aparece un número distinto de puntos (entre 1 y 6 puntos). Dados n dados y un valor M, quisieramos seleccionar una cara de cada uno de los dados de forma que el producto de los puntos que aparecen en las caras seleccionadas de todos los dados sea exactamente M.

Para ello debes diseñar un algoritmo de *búsqueda con retroceso* que devuelva como resultado una secuencia con los valores de la cara seleccionada de cada dado de forma que su producto sea M.

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 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*? Justifícalas.

Si os sobra tiempo, podríais intentar responder también 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 justifícalas).

2. Una variante del problema anterior no permite que en la solución pueda aparecer el mismo valor de un dado más de V veces. ¿Serías capaz de adaptar tu algoritmo para que resolviera dicho problema?

Algoritmos voraces
8 de abril de 2009 (Mañana)

Prueba individual

Nombre: **Grupo:**

Kruskal **40 puntos**

El algoritmo de Kruskal encuentra el árbol de recubrimiento de coste mínimo en un grafo no dirigido y ponderado. No obstante, resulta interesante considerar algunos casos especiales, pues podría ser que encontrar el árbol de recubrimiento de coste mínimo para ellos fuera más eficiente si se modificase el algoritmo de Kruskal. Dados los siguientes supuestos:

1. Todas las aristas del grafo pesan 1.
2. Todas las aristas del grafo tienen como peso uno de dos posibles valores.
3. El grafo es un grafo conexo en el que cada vértice tiene exactamente dos aristas y todas las aristas forman un ciclo hamiltoniano.
4. Las aristas del grafo tienen pesos enteros en el rango $[0..|V|]$.

Indica qué modificaciones del algoritmo de Kruskal harías *en cada supuesto* para solucionar estos problemas con la mayor eficiencia computacional. Señala también, justificándolo, el coste temporal de las propuestas.

Algoritmos voraces
8 de abril de 2009 (Mañana)

Prueba colectiva

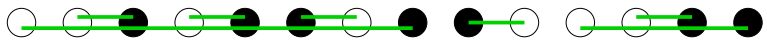
Grupo:
Asistentes:

Análisis de la solución voraz de un problema — 60 puntos

En una línea recta hay n puntos blancos y n puntos negros distribuidos equiespaciadamente: la distancia entre puntos consecutivos siempre es de un metro.

Deseamos conectar con cable cada punto negro con un punto blanco, *gastando la menor cantidad posible de cable*. La cantidad de cable necesario para empalmar dos puntos es igual a la distancia que los separa. Se nos ha ocurrido un algoritmo voraz para resolver el problema al que denominamos "Primero los más cercanos": Se divide el conjunto de puntos en "vivos" y "muertos". Inicialmente todos los puntos están vivos. Se ordena el conjunto de todas las posibles conexiones por valor no decreciente de distancia. Seleccionamos la primera posible conexión (la que consume menor cantidad de cable) y nos aseguramos de que une puntos vivos. Si es así, conectamos ambos puntos y los declaramos muertos. Iteramos el procedimiento hasta que todos los puntos estén muertos.

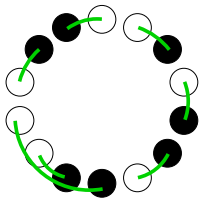
La figura muestra una conexión de puntos blancos con negros que se obtiene de aplicar el método propuesto sobre la figura anterior:



Las conexiones utilizan $1 + 1 + 1 + 1 + 1 + 7 + 3 = 15$ metros de cable.

a) Justificad si el algoritmo propuesto es correcto (*proporciona siempre la solución óptima*) y señalar, razonándolo, el coste temporal del mismo.

b) Suponiendo que los puntos se distribuyeran en un círculo tal y como se muestra en la siguiente figura (y manteniendo el resto de condiciones anteriores), un resultado posible de la aplicación del anterior algoritmo sería el siguiente:



¿Sería ahora correcto el método descrito? Justificadlo

Algoritmos voraces
8 de abril de 2009 (Tarde)

Prueba individual

Nombre: **Grupo:**

Búsqueda del camino más corto entre dos vértices **40 puntos**

Dado un grafo dirigido $G = (V, E)$ y ponderado por una función $d : E \to \mathbb{R}^{\geq 0}$, deseamos calcular de *la manera más eficiente posible* el camino más corto entre un vértice s y otro t. Sabemos que dicho camino se puede calcular haciendo uso de un conocido algoritmo (que presenta algunas variantes). Ante determinados supuestos puede que fuera más eficiente utilizar una estrategia alternativa o emplear una variante concreta del algoritmo. Dados los siguientes supuestos:

1. El grafo consta de una serie de vértices numerados de forma consecutiva (entre 0 y $|V|-1$), las únicas aristas que hay unen vértices consecutivos, $E = \{(i, i+1) \mid 0 \leq i < |V|\}$, y los vértices origen y destino del camino cumplen $s < t$.

2. El grafo es un árbol dirigido implementado mediante la representacion con referencias a padres (recordatorio: mediante un diccionario con una entrada por vértice en la que se almacena como valor el padre de dicho vértice en el árbol) y el vértice origen del camino s cumple que es un ascendente del vértice destino t.

3. El grafo tiene la particularidad de que de cada vértice salen aristas a todos los demás.

4. El grafo tiene grado de salida 3.

Debes describir brevemente qué harías *en cada supuesto* para obtener de la forma más eficiente posible el camino más corto (estrategia que aplicarías, algoritmo o variante que escogerías, modificación que harías sobre algún algoritmo...), así como justificar el coste temporal en función de $|V|$ y $|E|$ que conlleva la aplicación de la estrategia elegida.

Algoritmos voraces
8 de abril de 2009 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Análisis de la solución voraz de un problema **60 puntos**

Hemos montado una agencia de contactos personales (heterosexuales). Tenemos N clientes masculinos y M clientes femeninas a los que hemos pedido que cuantifiquen su interés con un número entre 0 (ningún interés) y 10 (mucho interés) por fijar una cita con cada una de las personas del otro sexo. El valor $c_{a,b}$ es el interés manifestado por la persona a en fijar una cita con la persona b.

Te presentamos un ejemplo suponiendo que hay $N = 3$ hombres y $M = 2$ mujeres. La matriz de la izquierda muestra el interés que cada hombre ha manifestado por mantener una cita con cada mujer y la matriz de la derecha muestra el interés que cada mujer ha manifestado por cada hombre:

	María	Nerea
Antonio	7	10
Benito	0	1
Carlos	3	0

	Antonio	Benito	Carlos
María	10	0	7
Nerea	6	8	1

El índice de compatibilidad de una pareja (a, b) se define como el producto de $c_{a,b}$ por $c_{b,a}$ y es, evidentemente, un valor numérico entre 0 y 100. En el ejemplo, el índice de compatibilidad entre Antonio y María es 70 ($= 7 \cdot 10$) y el índice de compatibilidad entre Benito y Nerea es 8 ($= 1 \cdot 8$).

La empresa cobra en función de cada cita organizada. El importe facturado por cita es una cantidad de euros igual al índice de compatibilidad de las personas convocadas. Acuciados por las deudas, hemos decidido organizar K citas en el mismo día, por lo que no podemos convocar a una misma persona a dos citas. Para maximizar el beneficio decidimos aplicar el siguiente procedimiento voraz y aplicarlo K veces: citar al par de personas de mayor compatibilidad y eliminar a ambas de la lista de personas a las que podemos citar. En el ejemplo, con $K = 2$, si citamos a Antonio y María por una parte ($7 \cdot 10 = 70$ euros) y, por otra, a Benito y Nerea ($1 \cdot 8 = 8$ euros), ganamos un total de 78 euros.

a) ¿Qué coste temporal presenta ese algoritmo?

b) ¿Encuentra siempre ese algoritmo la solución óptima, es decir, encuentra las K citas (sin repetir persona) que mayor beneficio nos reportan?

Cuando ya hemos saneado la situación financiera, trabajamos con algo más de calma y organizamos una nueva serie de K citas. Como ya no tenemos la restricción de que todas sean en el mismo día, planificamos estas nuevas K citas con la misma estrategia voraz (escoger cada vez la pareja con mayor compatibilidad), pero con la posibilidad de que una misma persona participe en más de una cita. En el ejemplo, también con $K = 2$, podemos citar a Antonio y María por una parte ($7 \cdot 10 = 70$ euros) y a Antonio y Nerea por otra ($10 \times 6 = 60$ euros), con un beneficio total de 130 euros.

c) ¿Qué coste temporal presenta ese algoritmo?

d) ¿Encuentra siempre ese algoritmo la solución óptima, es decir, encuentra las K citas (con la posibilidad de repetir persona, pero no pareja) que mayor beneficio nos reportan?

Debéis razonar con claridad y concisión las respuestas.

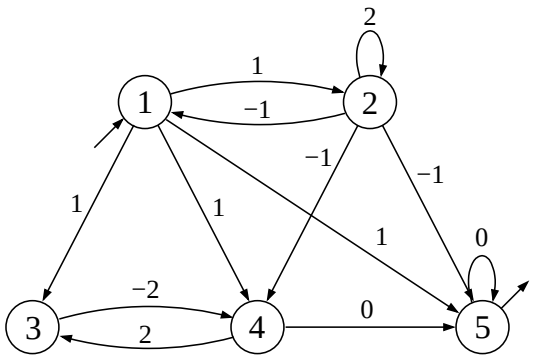
Programación dinámica
4 de mayo de 2009 (Mañana)

Prueba individual

Nombre: **Grupo:**

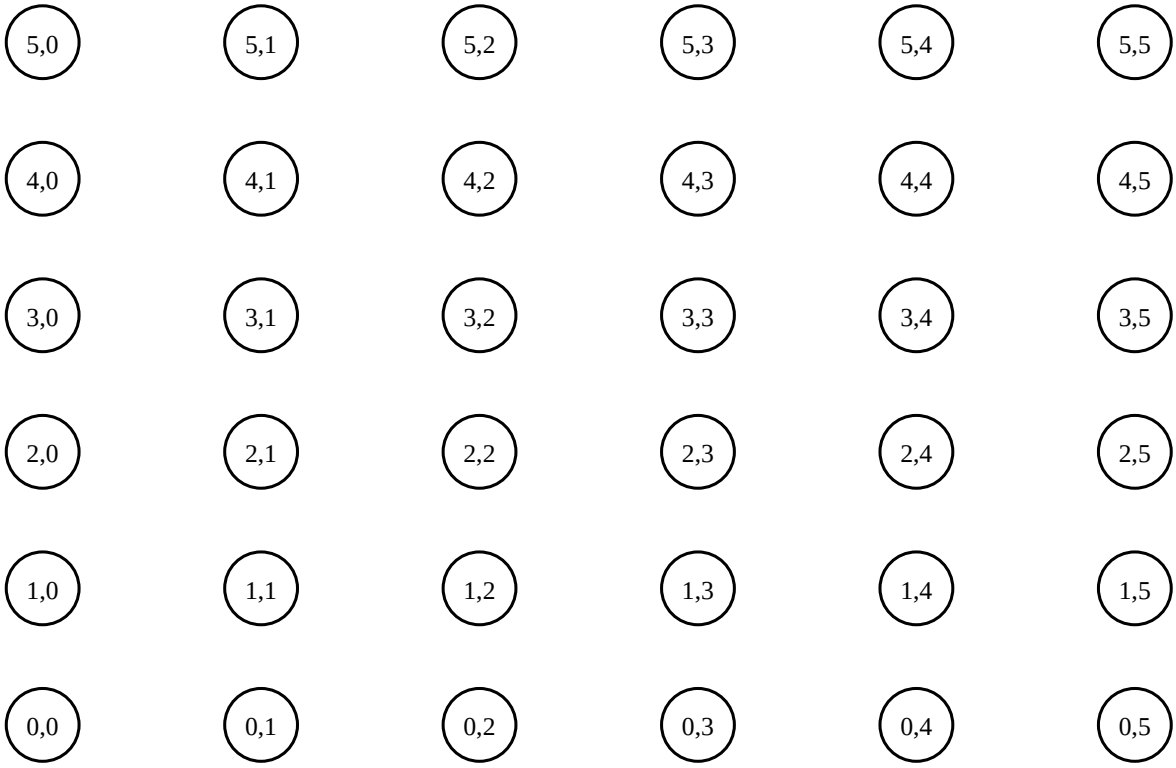
Problema del camino más corto formado por k aristas en un grafo **30 puntos**

Debes encontrar cuál es el camino más corto que empieza en el vértice $s = 1$ y acaba en el vértice $t = 5$ formado por exactamente $k = 3$ aristas en el siguiente grafo:



Para ello, se pide:

a) Dibuja, aprovechando la siguiente figura, el *grafo multietapa de dependencias* asociado al grafo anterior y a la ecuación recursiva (7.8) (pág. 7-37) que resolvía el problema. (Utiliza los nodos que necesites, tachando el resto, y traza los arcos entre estados correspondientes al problema propuesto):



b) Haz una traza del algoritmo `k_edges_shortest_distance` (pág 7-38). Para ello, escribe arriba de cada uno de los estados del grafo del apartado anterior el valor que el algoritmo almacena en el diccionario `D`.

c) ¿Cuál es el valor de la solución óptima? ¿En qué estado se encuentra? ¿Cuál sería la secuencia de aristas del grafo original que devolvería un algoritmo que calculara la solución óptima del problema (si hay más de una secuencia óptima, indica una cualquiera)?

Programación dinámica
4 de mayo de 2009 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema de la mochila con exactamente M objetos — 40 puntos

La siguiente ecuación recursiva permite plantear la resolución del problema de la mochila discreta cuando deseamos obtener el máximo beneficio (sin exceder la capacidad de carga W) cargando exactamente M objetos de los N disponibles:

$$V(i,c,m) = \begin{cases} 0, & \text{si } m = 0; \\ -\infty, & \text{si } m > 0 \text{ e } (i = 0 \text{ o } c = 0); \\ \max\left(V(i-1,c,m), V(i-1,c-w_i,m-1)+v_i\right), & \text{si } m > 0,\ i > 0,\ c > 0 \text{ y } w_i \leq c; \\ V(i-1,c,m), & \text{si } m > 0,\ i > 0,\ c > 0 \text{ y } w_i > c. \end{cases}$$

donde la llamada $V(N,W,M)$ proporciona el valor de la solución óptima.

Debéis escribir el algoritmo recursivo con memorización asociado a la anterior ecuación, justificando su coste temporal y espacial.

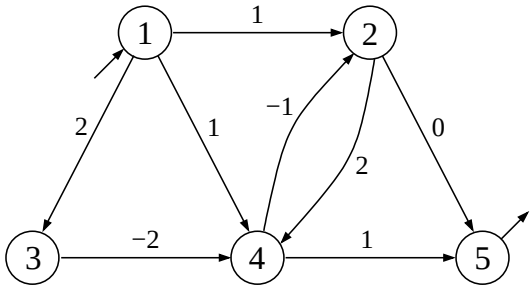
Programación dinámica
4 de mayo de 2009 (Tarde)

Prueba individual

Nombre: **Grupo:**

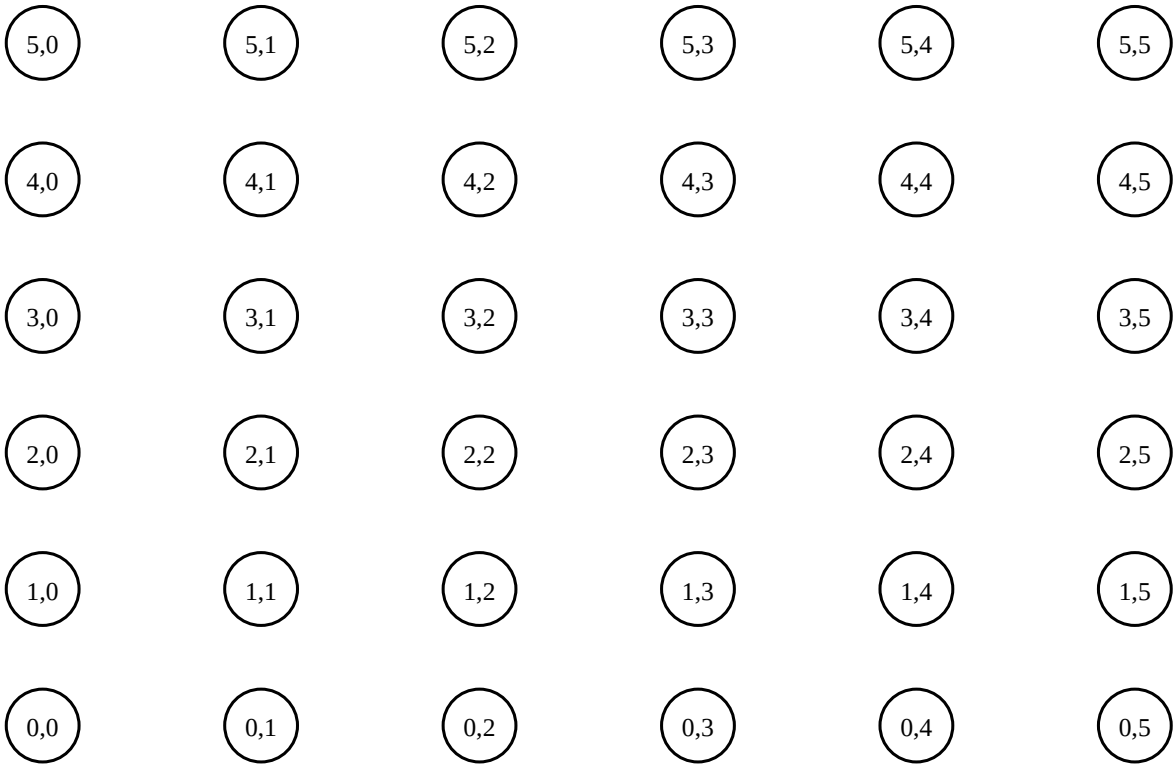
Problema del camino más corto en un grafo sin ciclos negativos: Bellman-Ford **30 puntos**

Debes encontrar cuál es el camino más corto que empieza en el vértice $s = 1$ y acaba en el vértice $t = 5$ en el siguiente grafo que contiene ciclos no negativos:



Para ello, se pide:

a) Dibuja, aprovechando la siguiente figura, el *grafo multietapa de dependencias* asociado a la resolución del problema que se propone en la pág. 7-39 para el ejemplo anterior. (Utiliza los nodos que necesites, tachando el resto, y traza los arcos entre estados correspondientes al problema propuesto):



b) Haz una traza del algoritmo de Bellman-Ford `shortest_distance` (pág 7-40). Para ello, escribe arriba de cada uno de los estados del grafo del apartado anterior el valor que el algoritmo almacena en el diccionario `D`.

c) ¿Cuál es el valor de la solución óptima? ¿En qué estado se encuentra? ¿Cuál sería la secuencia de aristas del grafo original que devolvería un algoritmo que calculara la solución óptima del problema (si hay más de una secuencia óptima, indica una cualquiera)?

Programación dinámica
4 de mayo de 2009 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema de la mochila con un número ilimitado de objetos de cada valor **40 puntos**

La siguiente ecuación recursiva permite plantear una resolución del problema de la mochila discreta cuando deseamos obtener el máximo beneficio (sin exceder la capacidad de carga W a partir de N tipos de objetos diferentes) cuando disponemos de un número ilimitado de objetos de cada valor v_i y peso w_i ($1 \leq i \leq N$):

$$V(i,c) = \begin{cases} 0, & \text{si } i = 0 \text{ o } c = 0; \\ \max\limits_{0 \leq x \leq \lfloor c/w_i \rfloor} \left(V(i-1, c - w_i * x) + v_i * x \right), & \text{si } i > 0 \text{ y } c > 0. \end{cases}$$

donde la llamada $V(N,W)$ proporciona el valor de la solución óptima.

a) Dibujad, aprovechando la siguiente cuadrícula, el *grafo multietapa de dependencias* asociado a la anterior ecuación recursiva para los datos $W = 6$, $N = 5$, $v = [90, 75, 60, 20, 10]$ y $w = [4,3,3,2,2]$:

(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)
(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) Escribid el algoritmo iterativo que permite conocer el valor de la solución óptima para la ecuación recursiva propuesta, justificando su coste temporal y espacial.

Programación dinámica
13 de mayo de 2009 (Mañana)

Prueba individual

Nombre: **Grupo:**

Problema **50 puntos**

Un vendedor ambulante de productos de artesanía realiza una ruta a través de una serie de P pueblos dispuestos a lo largo de una carretera. Parte de la ciudad en la que habita (situada al comienzo de la carretera) y debe llegar a una ciudad que se encuentra en el otro extremo de la carretera sin retroceder nunca. Su sistema de ventas es siempre el mismo: un día llega a un pueblo por la mañana, vende toda la mercancía, compra la artesanía típica del pueblo y continúa hasta otro pueblo, en el que, al día siguiente, repetirá la operación. Al principio del viaje partirá con productos típicos de su ciudad y al acabar deberá vender la mercancía que le quede en la ciudad final de trayecto. El comerciante, una vez ha parado en un pueblo, no vuelve a detenerse hasta haber atravesado un mínimo de dos pueblos más (al ser pueblos vecinos no podría vender a buen precio los objetos).

El comerciante tiene una estimación del beneficio que puede obtener en un determinado pueblo en función de la mercancía que venda en él. Es decir, si numeramos los pueblos entre 1 y P según el orden en que aparecen en la carretera, el beneficio que obtiene al vender la mercancia de un pueblo i en otro j sería, $b(i, j)$, para $1 \leq i < j \leq P$.

Se pretende averiguar en que pueblos debería parar para obtener el máximo beneficio posible.

La siguiente ecuación recursiva devuelve el valor del máximo beneficio que puede obtener el comerciante en una ruta entre el pueblo 1 y el pueblo p:

$$B(p) = \begin{cases} 0, & \text{si } p = 1; \\ -\infty, & \text{si } p = 2 \text{ o } p = 3; \\ \max\limits_{1 \leq p' \leq p-3} B(p') + b(p', p), & \text{si } p > 3. \end{cases}$$

donde la llamada $B(P)$ proporciona el máximo beneficio para la ruta completa entre 1 y P. Se pide:

a) Escribe el algoritmo recursivo con memorización asociado a la anterior ecuación.

b) Representa el grafo de dependencias asociado a la ecuación recursiva para un problema con $P = 10$ pueblos.

c) Indica cuál sería el coste temporal y espacial de un algoritmo iterativo diseñado a partir de la ecuación presentada. Justifica brevemente tu respuesta.

Programación dinámica
13 de mayo de 2009 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

Sea el anterior problema:

****

Un vendedor ambulante de productos de artesanía realiza una ruta a través de una serie de P pueblos dispuestos a lo largo de una carretera. Parte de la ciudad en la que habita (situada al comienzo de la carretera) y debe llegar a una ciudad que se encuentra en el otro extremo de la carretera sin retroceder nunca. Su sistema de ventas es siempre el mismo: un día llega a un pueblo por la mañana, vende toda la mercancía, compra la artesanía típica del pueblo y continúa hasta otro pueblo, en el que, al día siguiente, repetirá la operación. Al principio del viaje partirá con productos típicos de su ciudad y al acabar deberá vender la mercancía que le quede en la ciudad final de trayecto. El comerciante, una vez ha parado en un pueblo, no vuelve a detenerse hasta haber atravesado un mínimo de dos pueblos más (al ser pueblos vecinos no podría vender a buen precio los objetos).

El comerciante tiene una estimación del beneficio que puede obtener en un determinado pueblo en función de la mercancía que venda en él. Es decir, si numeramos los pueblos entre 1 y P según el orden en que aparecen en la carretera, el beneficio que obtiene al vender la mercancia de un pueblo i en otro j sería, $b(i, j)$, para $1 \leq i < j \leq P$.

Se pretende averiguar en que pueblos debería parar para obtener el máximo beneficio posible.

La siguiente ecuación recursiva devuelve el valor del máximo beneficio que puede obtener el comerciante en una ruta entre el pueblo 1 y el pueblo p:

$$B(p) = \begin{cases} 0, & \text{si } p = 1; \\ -\infty, & \text{si } p = 2 \text{ o } p = 3; \\ \max\limits_{1 \leq p' \leq p-3} B(p') + b(p', p), & \text{si } p > 3. \end{cases}$$

donde la llamada $B(P)$ proporciona el máximo beneficio para la ruta completa entre 1 y P.

****

Sobre el problema original se pide:

a) Escribid el algoritmo iterativo *que proporciona la secuencia de decisiones óptima, concretamente, debe proporcionar los pueblos del recorrido* que permite obtener el máximo beneficio al comerciante.

A continuación, se proponen las siguientes modificaciones en el enunciado del problema del vendedor ambulante:

1. El vendedor debe hacer la ruta en D días (es decir, deberá pasar exactamente por D pueblos, sin contar el pueblo de origen).
2. Que la mercancia de un pueblo no se pueda vender en los dos siguientes porque no da beneficio es un criterio muy general. El vendedor lo ha mejorado y ahora, para cada pueblo p, sabe cuál es el pueblo anterior más cercano, $v[p]$, cuya venta de mercancia en p ya proporciona beneficio. Por ejemplo, $v[7] = 5$ implica que en el pueblo 7 podemos vender con beneficio la mercancia de pueblo 5 y de todos los anteriores al 5. Si $v[p] = 0$ en el pueblo p no podemos vender la mercancia de ningún pueblo anterior.

El resto de condiciones se mantienen.

Para este *nuevo problema* se pide:

a) Plantead la ecuación recursiva de programación dinámica que calcule el máximo beneficio que se puede obtener con cualquier secuencia válida de pueblos, indicando cuál debe ser la llamada inicial que se efectúe a dicha función para obtener el valor óptimo.

b) Representad el grafo de dependencias entre las llamadas de la ecuación recursiva para $P = 6$ pueblos, $D = 3$ días y $v[1] = 0$, $v[2] = 0$, $v[3] = 1$, $v[4] = 3$, $v[5] = 3$ y $v[6] = 4$.

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.

Programación dinámica
13 de mayo de 2009 (Tarde)

Prueba individual

Nombre: **Grupo:**

Problema **50 puntos**

Dados los números naturales N y K, una descomposición de N en K factores positivos es una secuencia n_1, n_2, $\ldots$, n_K, tal que

$$N = \prod_{1 \leq k \leq K} n_k \,.$$

Por ejemplo, si $N = 12$ y $K = 3$ tenemos las siguientes cuatro factorizaciones posibles: $12 \cdot 1 \cdot 1$, $6 \cdot 2 \cdot 1$, $3 \cdot 2 \cdot 2$ y $3 \cdot 4 \cdot 1$.

Deseamos obtener la descomposición de N en K factores de modo que la suma de los factores,

$$\sum_{1 \leq k \leq K} n_k,$$

sea la *mínima* posible. Siguiendo con el ejemplo anterior, la suma de los factores es, respectivamente, $12+1+1 = 14$, $6 + 2 + 1 = 9$, $3 + 2 + 2 = 7$ y $3 + 4 + 1 = 8$, con lo que la descomposición óptima es $3, 2, 2$.

La siguiente ecuación recursiva devuelve el valor de la mínima suma de factores que se puede obtener de la descomposición de n en k factores:

$$S(n,k) = \begin{cases} n, & \text{si } k = 1; \\ k, & \text{si } k > 1 \text{ y } n = 1; \\ \min\limits_{\substack{1 \leq m \leq n: \\ (n \text{ mód } m) = 0}} S(n/m, k-1) + m, & \text{si } k > 1 \text{ y } n > 1, \end{cases}$$

donde la llamada $S(N, K)$ proporciona dicho valor mínimo para la descomposición del número N en K factores. Se pide:

a) Escribe el algoritmo recursivo con memorización asociado a la anterior ecuación.

b) Representa el grafo de dependencias asociado a la ecuación recursiva para $N = 12$ y $K = 3$.

c) Indica cuál sería el coste temporal y espacial de un algoritmo iterativo diseñado a partir de la ecuación presentada. Justifica brevemente tu respuesta.

Programación dinámica
13 de mayo de 2009 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

Sea el anterior problema:

****

Dados los números naturales N y K, una descomposición de N en K factores positivos es una secuencia $n_1, n_2, \ldots, n_K$, tal que

$$N = \prod_{1 \leq k \leq K} n_k \,.$$

Por ejemplo, si $N = 12$ y $K = 3$ tenemos las siguientes cuatro factorizaciones posibles: $12 \cdot 1 \cdot 1$, $6 \cdot 2 \cdot 1$, $3 \cdot 2 \cdot 2$ y $3 \cdot 4 \cdot 1$.

Deseamos obtener la descomposición de N en K factores de modo que la suma de los factores,

$$\sum_{1 \leq k \leq K} n_k,$$

sea la *mínima* posible. Siguiendo con el ejemplo anterior, la suma de los factores es, respectivamente, $12 + 1 + 1 = 14$, $6 + 2 + 1 = 9$, $3 + 2 + 2 = 7$ y $3 + 4 + 1 = 8$, con lo que la descomposición óptima es $3, 2, 2$.

La siguiente ecuación recursiva devuelve el valor de la mínima suma de factores que se puede obtener de la descomposición de n en k factores:

$$S(n,k) = \begin{cases} n, & \text{si } k = 1; \\ k, & \text{si } k > 1 \text{ y } n = 1; \\ \min\limits_{\substack{1 \leq m \leq n: \\ (n \bmod m) = 0}} S(n/m, k-1) + m, & \text{si } k > 1 \text{ y } n > 1, \end{cases}$$

donde la llamada $S(N, K)$ proporciona dicho valor mínimo para la descomposición del número N en K factores.

****

Sobre el problema original se pide:

a) Escribid el algoritmo iterativo *con reducción de la complejidad espacial* que permite obtener el valor de la suma de la descomposición óptima.

A continuación, se proponen las siguientes modificaciones en el enunciado del problema de la minimización de la suma de factores resultantes de la descomposición de un número:

1. La descomposición no tiene por qué tener un número fijo K de componentes.
2. Los factores tienen que ser números mayores que 1.

El resto de condiciones se mantienen.
Con las nuevas condiciones y el ejemplo $N = 12$, las siguientes descomposiciones serían válidas: $3 \cdot 2 \cdot 2$, $6 \cdot 2$, $3 \cdot 4$, mientras que la descomposición $12 \cdot 1 \cdot 1$ no lo sería.
Para este *nuevo problema* se pide:

a) Plantead la ecuación recursiva de programación dinámica que calcule la suma de los factores asociados a la descomposición óptima, indicando cuál debe ser la llamada que se efectue a dicha función para obtener el valor óptimo.

b) Representad el grafo de dependencias entre las llamadas de la ecuación recursiva para $N = 12$.

c) Indicad cuál sería el coste temporal y espacial de un algoritmo iterativo diseñado a partir de vuestra ecuación recursiva que devolviera *la descomposición óptima*. Justificad brevemente la respuesta.

d) ¿Sería posible reducir el coste espacial indicado en el apartado anterior si hubiera que diseñar un algoritmo iterativo que sólo encontrara *el valor* de la suma de la descomposición óptima? Si es así, indicad cuál sería dicho coste y justificadlo.

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

Primer control parcial de Esquemas Algorítmicos (IG24)

28 de marzo de 2009

1. Divide y vencerás — 5 puntos

Sean un vector a, que contiene n elementos sin repetir ordenados de forma creciente, y un vector b que contiene m elementos (muchos de ellos repetidos) también ordenados de forma creciente. El número de elementos de b es mucho mayor que el de a ($m >>> n$). Se desea conocer, para cada elemento contenido en a, cuantas veces aparece en b. El siguiente ejemplo ilustra una instancia del problema: dados $a = [3, 5, 8, 10]$ y $b = [0, 0, 0, 1, 1, 3, 3, 3, 3, 3, 5, 7, 7, 7, 7, 7, 10, 10, 10, 10]$, la solución sería que el 3 aparece cinco veces, el 5 aparece una vez, el 8 ninguna y el 10 cuatro. Se pide:

a) Diseña un algoritmo que, dados a y b, resuelva el problema mediante la técnica recursiva de *divide y vencerás*. El algoritmo debe ser lo más eficiente posible y, en cualquier caso, el coste temporal del mismo debe ser inferior a $O(nm)$.

   Fijate que lo que se pide es que la *estrategia fundamental* con la que resuelvas el problema sea la de divide y vencerás, no que el algoritmo completo tenga que ser exclusivamente 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 desde el algoritmo principal para devolver los valores pedidos. En cualquier caso, no olvides incluir todo el código necesario para la resolución completa del problema.

b) Indica si el algoritmo presenta un comportamiento uniforme o existe un mejor y un peor caso. Si el comportamiento es uniforme, trata de modificarlo para que ante determinadas instancias pueda finalizar antes su ejecución. Señala en qué condiciones (ante que tipos de entradas) puede el algoritmo comportarse en el mejor de los casos y en qué condiciones en el peor.

c) Realiza un análisis de la complejidad temporal y espacial del algoritmo presentado, justificando los costes en el mejor y en el peor de los casos.

d) Si el algoritmo propuesto (o alguna de sus funciones) presenta recursividad por cola, elimínala e indica si alguno de los costes calculados en el apartado anterior varía.

2. Búsqueda con retroceso — 5 puntos

Estamos en un país exotico y queremos enviar una postal a un amigo. Para ello, hemos adquirido un lote de sellos de n valores diferentes y que contiene 3 sellos de cada valor. En la oficina de correos nos han indicado que enviar una postal a nuestro amigo tiene un coste C. Además, nos informan de que en las postales del país es obligatorio colocar un número exacto de sellos M, que depende del tipo de postal.

Dados los valores de los sellos del lote, $(v_1, v_2, \ldots, v_n)$, C y M, queremos encontrar una combinación de exactamente M sellos de coste total C, sabiendo que no podemos usar más de tres sellos del mismo valor.

Por ejemplo, si los valores fuesen $v = [1, 2, 3, 5, 8, 10]$, el coste del envío $C = 16$ y tuviese que emplear $M = 5$ sellos, una posible solución podría ser usar un sello de 10, dos de 2 y dos de 1. No serían soluciones válidas usar un sello de 10, otro de 5 y otro de 1 (no son 5 sellos), ni usar uno de 8 y cuatro de 2 (se usan más de tres sellos de un valor).

Queremos diseñar un algoritmo de *búsqueda con retroceso* que resuelva el problema de encontrar una combinación de sellos que cumpla los requisitos indicados.

Se pide que:

a) Describas cómo vas a representar los estados s 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).

Segundo control parcial de Esquemas Algorítmicos (IG24)

16 de mayo de 2009

1. Algoritmos voraces — 3.25 puntos

En una gran fábrica de vehículos las diferentes secciones de la fábrica se hayan distribuidas en distintos edificios. Cuando una sección necesita envíar material a alguna otra (piezas, herramientas,...), utiliza a algún empleado que debe dejar su trabajo para hacer la entrega, con los consiguientes retrasos en la fabricación.

Para evitarlo, la fábrica ha decidido poner en marcha un sistema automatizado de transporte de material, empleando una vagoneta que se traslada por raíles entre las distintas secciones. Quisieran gastarse la menor cantidad de dinero posible en el sistema, cuyo coste depende, básicamente, de la cantidad de raíles que haya que instalar. Como conectar cada sección con todas los demás sería muy costoso, han pensado que cada sección se pueda utilizar como punto intermedio para el traslado de material, de manera que si, por ejemplo, la sección a tiene que enviar material a la sección b no es necesario que haya una vía directa entre las dos, sino que si existe una vía que une a la sección a con la c, una que une la c con la d y otra entre la d y la b, se podría seguir ese trayecto para enviar el material desde a hasta b.

La fábrica está estudiando algunas propuestas de *estrategias voraces* que permiten determinar, a partir del coste que supone colocar raíles entre todos los pares de secciones, $r(a, b)$, *qué vías son las que habría que instalar de forma que el coste de la instalación sea el mínimo* posible garantizando que se van a poder hacer envíos entre cualquier par de secciones. He aquí tres de ellas:

1. Para cada sección estudiamos cuál es la sección con la que le cuesta menos construir un raíl y construimos esos raíles.
2. Partiendo de una sección i al azar, estudiamos con cuál podemos poner una vía lo más barata posible j, y construimos el raíl. A continuación nos desplazamos a j y realizamos el mismo estudio, excluyendo a i, seleccionamos la sección k a la que podemos llegar con menor coste y construimos el raíl entre j y k. Hacemos lo mismo con k (excluyendo los raíles que puedan ir hasta i y hasta j) y así sucesivamente hasta haber unido todas las secciones.
3. Como lo que nos interesa es construir las uniones entre secciones más baratas, lo que haremos es construir en primer lugar la unión más barata de todas, luego la segunda más barata, y continuaremos de esta manera hasta que veamos que no queda ninguna sección sin estar unida, al menos, a otra sección.

Se pide:

a) 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.

b) 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).

c) 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).

d) Si de entre las estrategias propuestas has detectado más de una que garantice la obtención de la solución óptima, di cuál emplearías. Si no es así y crees que puede haber alguna estrategia correcta alternativa a las propuestas, coméntala, señalándo también su coste (en el caso en que hubiera más de una, cita únicamente la mejor).

2. Programación dinámica — 4 puntos

Una empresa que tiene que promocionar un nuevo producto ha decidido, como parte de la campaña publicitaria, desplazar una furgoneta de la compañía a distintas zonas de la ciudad con muestras para regalar. La campaña durará D días. La furgoneta debe ir cada día a una única zona de la ciudad (la ciudad se ha dividido para la campaña en Z zonas, numeradas, por comodidad, entre 1 y Z). La furgoneta no tiene porqué visitar todas las zonas y puede ir más de una vez a una zona; sin embargo, si un día visita una zona, al día siguiente no puede volver a esa ni a las zonas vecinas (para cada zona z ($1 \leq z \leq Z$), $v(z)$ nos indica cuáles son sus zonas vecinas). Eso sí, la campaña ha de comenzar y acabar en la zona 1.

La promoción del producto que obtiene la compañía en una visita de la furgoneta a la zona z el día d de la campaña ha sido cuantificada en $p(z, d)$. La promoción total conseguida en la campaña es la suma de los valores de las promociones diarias.

Deseamos saber, aplicando la estratégia de *programación dinámica*, qué zonas debe visitar cada día la furgoneta para máximizar la promoción conseguida con la campaña.

1. Con las condiciones anteriormente expuestas:
   a) Formaliza el problema en términos de optimización.
   b) Plantea la ecuación recursiva asociada al hecho de visitar durante D días distintas zonas de la ciudad con las condiciones previamente indicadas para conseguir la máxima promoción del producto. Indica cuál debe ser la llamada que se efectue a dicha función para obtener el valor óptimo.

c) Representa el grafo de dependencias entre llamadas de la ecuación recursiva para el ejemplo $D = 4$, $Z = 5$ y $v[1] = \{2, 4\}$, $v[2] = \{1, 3\}$, $v[3] = \{2, 4\}$, $v[4] = \{1, 3, 5\}$, $v[5] = \{4\}$ (donde, por ejemplo, $v[1] = \{2, 4\}$ indica que la zona 1 tiene como vecinas a las zonas 2 y 4). 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 sólo nos interesara obtener el valor de la mejor promoción).

2. Una variante del problema sería la siguiente: la empresa se ha dado cuenta que no importa repetir la visita a una zona (o a zonas vecinas) de un día al siguiente, ya que eso permitiría reforzar la campaña en dicha zona. Por otra parte, tampoco es obligatorio empezar y acabar la campaña en una zona concreta.

   ¿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 — 2.75 puntos

Un joven periodista que acaba de ser contratado como becario en una cadena de televisión se ha trazado como objetivo llegar a ser el presentador estrella de la cadena en el menor tiempo posible. Sabe que los periodistas de la cadena se dividen en C categorías, numeradas jerárquicamente, de manera que a los becarios les corresponde la categoría 1 y al mejor presentador la C.

El sistema de promoción interna en la empresa es muy exigente, sólo se puede ascender a la categoría c como mucho desde la categoría $c - p(c)$, donde $p(c)$ devuelve, para cada categoría $2 \leq c \leq C$, el número máximo de categorías anteriores desde las que se puede ascender (si, por ejemplo, $c = 6$ y $p(6) = 2$, eso implica que a la categoría 6 se puede ascender desde la 5 y la $4 (= 6 - 2)$ pero no desde categorías previas). Por otra parte, para conseguir el ascenso a una categoría c desde una categoría anterior válida c', hay que lograr una serie de méritos que cuesta un tiempo $t(c', c)$ reunir. Finalmente, la empresa exige, para llegar a la máxima categoría, haber pasado en total por al menos M categorías (incluyendo la C).

Debes averiguar, empleando la técnica de programación dinámica, por qué categorías debe pasar el periodista, partiendo de becario, para llegar a ser la estrella de la cadena en el menor tiempo posible con las restricciones anteriores. Como ayuda, la siguiente ecuación recursiva de programación dinámica permite resolver el problema de obtener el tiempo mínimo necesario para conseguirlo:

$$T(c, m) = \begin{cases} 0, & \text{si } c = 1 \text{ y } m = 1; \\ +\infty, & \text{si } (c = 1 \text{ y } m > 1) \text{ o } (c > 1 \text{ y } m = 1); \\ \min\limits_{c - p(c) \leq c' < c} T(c', m - 1) + t(c', c), & \text{si } c > 1 \text{ y } m > 1. \end{cases}$$

La llamada $\min\limits_{M \leq m \leq C} T(C, m)$ proporciona dicho valor.

Se pide:

a) Representa el grafo de dependencias para el problema en el que $C = 6$, $M = 4$ y $p = [None, 1, 2, 1, 2, 3]$ (recuerda que $p[1]$ es $None$ porque la categoría de becario es la de entrada en la empresa). Indica qué representa el estado $(5, 3)$ y cómo interpretarías el significado de un arco entre los estados $(3, 2)$ y $(5, 3)$ dentro del grafo.

b) Diseña un algoritmo iterativo que devuelva *las categorías por las que debe pasar el periodista* hasta llegar a estrella empleando el menor tiempo posible.

c) Indica cuál es el coste temporal y espacial del algoritmo diseñado justificando brevemente tu respuesta. ¿Podrías acotar algo más alguno de los anteriores costes si nos dijeran que el valor máximo almacenado en p es 3? Justifícalo.

d) ¿Crees que sería posible reducir el coste espacial si únicamente nos interesará obtener el valor del mínimo tiempo necesario? Si es así, índica cuál sería dicho coste y justifícalo.

Examen convocatoria ordinaria de Esquemas Algorítmicos (IG24)

12 de junio de 2009

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

Sea un vector a que contiene n elementos, muchos de los cuales están repetidos. Los elementos repetidos se ubican en posiciones contiguas del vector. El número de elementos distintos que contiene a es m ($m <<< n$). Un ejemplo de vector podría ser el siguiente: a = [7, 7, 1, 1, 1, 6, 6, 6, 6, 6, 6, 6, 6, 12, 12, 12, 8, 9, 9, 9, 9, 9].

Has de diseñar un algoritmo de *divide y vencerás* que, dado un vector como el descrito, devuelva los elementos distintos contenidos en a (para el ejemplo anterior, el algoritmo devolvería {7, 1, 6, 12, 8, 9}). Se pìde:

a) Diseña un algoritmo recursivo *lo más eficiente posible* mediante la técnica de divide y vencerás que permita resolver este problema.

b) 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

Quisieramos resolver un pasatiempo de un periodico consistente en construir una frase de uso común (un refrán, una máxima, una frase célebre...) formada por M palabras que debemos elegir (y combinar) de entre un conjunto de N palabras posibles ($M < N$).

Con la idea de ayudarnos a resolver el problema con el ordenador nos proporcionan el número M de palabras que forman la frase, una lista l que contiene las N palabras que podemos utilizar y dos funciones, v_1 que nos permite saber si una secuencia (una lista) de palabras puede llegar a formar una frase con sentido, y v_2 que nos permite comprobar si una lista de palabras forma una frase válida (el coste de ejecutar v_1 y v_2 es lineal con el número de palabras que se le pasan).

Por ejemplo, dados $M = 5$ y la lista de palabras l = ['madrugar', 'hombres', 'mucho', 'no', 'todos', 'más', 'iguales', 'temprano', 'por', 'los', 'son', 'amanece'], nuestras funciones se comportarían de la siguiente manera:
v_1(['los', 'temprano','no']) = `False`,
v_1(['por', 'mucho', 'madrugar']) = `True`,
(en el primer caso la construcción no tiene sentido, mientras que en el segundo la construcción sintáctica es correcta de momento, podríamos llegar a tener una frase completa con sentido).
v_2(['por', 'mucho','madrugar']) = `False`,
v_2(['no', 'por', 'mucho', 'madrugar', 'amanece', 'más', 'temprano']) = `True`,
v_2(['todos', 'los', 'hombres', 'son', 'iguales']) = `True`, y
(en el primer caso la frase completa no tiene sentido, en el segundo y el tercero son válidas, pero sólo sería una solución la última, ya que es la única que contiene 5 palabras).

Debes desarrollar un algoritmo de *búsqueda con retroceso* que, dados M, l, v_1 y v_2, devuelva una frase válida de M carácteres. Se pide:

a) Describe cómo vas a representar los estados en la resolución del problema planteado.

b) Escribe un algoritmo de búsqueda con retroceso que resuelva el problema, indicando cuál sería la llamada inicial que realizarías.

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.

3. Voraces — 1,5 puntos

En un congreso organizado por la universidad tenemos que planificar la distribución en aulas de n conferencias (cada una de las cuales tiene una hora de comienzo y una hora de finalización) de forma que no haya más de una conferencia en la misma aula de forma simultanea. Queremos desarrollar un *algoritmo voraz* que realice una distribución de las conferencias entre aulas *haciendo uso del menor número posible de aulas*. 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 bjetivo que se pretende optimizar y qué debe cumplir una solución factible para ser considerada óptima.

b) Propón una estrategia voraz (descríbela con la mayor precisión posible) que devuelva la solución óptima para el problema ante cualquier entrada.

c) Indica y justifica el coste temporal y espacial de la estrategía (y del algoritmo) propuestos.

4. Programación dinámica **3,5 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 liquidación 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, $s(i)$.

Deseamos diseñar un algoritmo mediante la estrategia de *programación dinámica* que determine qué piezas hemos de emplear para costruir una valla con la longitud indicada con el menor coste posible. Se pide:

a) Formaliza el problema en términos de optimización.

b) Diseña una ecuación recursiva que calcule el mínimo coste necesario para construir los M metros de valla, 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 del mínimo coste empleando la menor ocupación espacial (en términos asintóticos) posible. Indica y justifica sus costes espacial y temporal.

Examen segunda convocatoria de Esquemas Algorítmicos (IG24)

1 de septiembre de 2009

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

Hay un juego infantil de azar en el que se van eliminando participantes en sucesivas etapas hasta que queda sólo uno (que es el ganador) y que funciona siguiendo una estrategia de divide y vencerás: Inicialmente, se situan los n participantes en una fila circular de manera que el primero pueda llegar a tocar al segundo, el segundo al tercero y así sucesivamente hasta el n-ésimo participante, que debe poder tocar al primero. En cada etapa se sigue el siguiente procedimiento: se elige al azar a un participante; dicho participante le da una patada en el culo al participante que va tras el, que queda eliminado; a continuación, el siguiente partipante en la fila (el que está detras del que acaban de eliminar) le da una patada al que va tras el, eliminándolo, y así hasta llegar al participante anterior al elegido en el sorteo que, en caso de no haber recibido una patada, se la da al primero, eliminándolo; entonces, sin modificar el orden de los participantes en la fila, se estrecha el círculo para que puedan alcanzarse los participantes y se inicia una nueva etapa (sorteo, patada y eliminación, estrechamiento del círculo), hasta que sólo queda el ganador.

Suponiendo que recibimos un vector a con los nombres de los n participantes iniciales en el orden en que se colocarán en la fila, se pide:

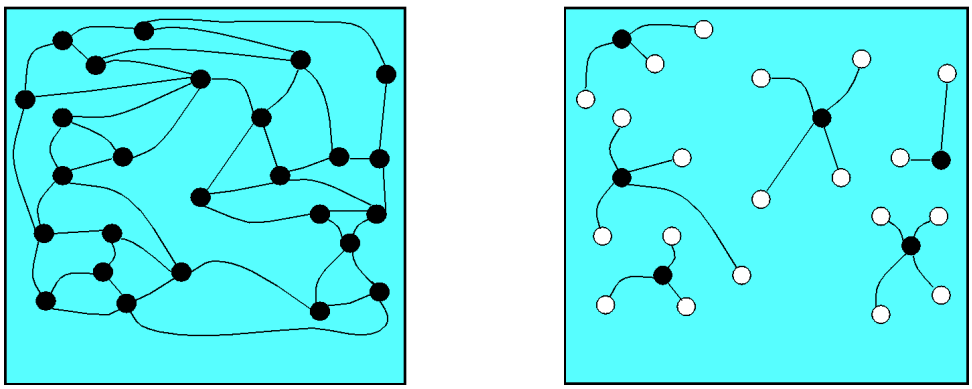
a) Implementa un algoritmo recursivo *mediante la técnica de divide y vencerás* que siga la estratégia anteriormente descrita y que devuelva el nombre del ganador. (Para efectuar los sorteos tienes disponibles la función de python `randrange(m)` (devuelve un valor entero aleatorio entre 0 y m-1) o la función `sample(lista, 1)` (devuelve un vector con un elemento seleccionado al azar de `lista`).)

b) Analiza y justifica la complejidad temporal y espacial del algoritmo desarrollado en el apartado anterior, indicando si presenta un comportamiento uniforme o existe mejor y peor caso.

c) Si el algoritmo propuesto presenta recursividad por cola, elimínala e indica si alguno de los costes calculados en el apartado anterior varía.

2. Búsqueda con retroceso — 2,5 puntos

Un *conjunto dominante* de m vértices en un grafo no dirigido es un subconjunto formado por m vértices del grafo que cumplen que, considerándolos a ellos y a todos los vértices que son adyacentes a ellos, se tienen todos los vértices del grafo. La siguiente figura ilustra este concepto:



(a) (b)

donde (a) es un ejemplo de grafo no dirigido y (b) un conjunto dominante de 6 vértices sobre dicho grafo en el que los vértices sombreados son los que forman el conjunto dominante y los no sombreados son los adyacentes a estos (como puede observarse, entre los sombreados y los no sombreados están considerados todos los vértices del grafo original).

Dado un grafo no dirigido $G = (V, E)$ y un valor m, diseña un algoritmo de *búsqueda con retroceso* que encuentre un conjunto dominante compuesto por exactamente m vértices. 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: 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 qué 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*? Justifícalas.

d) ¿Crees que es posible mejorar la eficiencia del algoritmo que has desarrollado introduciendo alguna variable o estructura de datos adicional o haciendo algún tipo de preproceso? Si es así, describe con claridad la mejora o mejoras que introducirías en el algoritmo e indica si ello afectaría a los costes previamente calculados.

3. Voraces **1,5 puntos**

Tenemos que recoger la cosecha en n campos, cada uno con un cultivo diferente. La recolección de un campo exige un día completo de trabajo. Para cada campo i ($1 \leq i \leq n$) conocemos el beneficio $b[i]$ que nos reportará la venta de su cosecha, así como el número de días máximo $d[i]$ que, a partir de hoy, tardaría en estropearse el cultivo si no lo cosechamos.

Queremos desarrollar un *algoritmo voraz* que indique qué campos y en qué días debemos recolectar cada uno de ellos para maximizar el beneficio total obtenido. 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) Propón una estrategia voraz (descríbela con precisión) que devuelva la solución óptima para el problema ante cualquier entrada.

c) Indica y justifica el coste temporal y espacial de la estrategía propuesta.

d) ¿La estrategia que has desarrollado permite resolver también el problema si la recolección de cada campo requiriera un número variable de días, dependiendo de cada uno de los campos? Justifícalo adecuadamente.

4. Programación dinámica **3,5 puntos**

Disponemos de una serie de R recursos, $r_1, r_2, \ldots, r_R$, que podemos asignar a $d_1, d_2, \ldots, d_D$ actividades, con $R \geq D$. Podemos asignar una serie de recursos *contiguos* a una misma actividad. Si asignamos los recursos $r_i, r_{i+1}, \ldots, r_j$ a la actividad d_l, asignaremos los recursos $r_{j+1}, r_{j+2}, \ldots, r_k$ a la actividad d_{l+1}, donde $i \leq j < k$. No podemos dejar recurso alguno sin asignar a alguna actividad y cada actividad ha de recibir al menos uno de los recursos. La asignación de $r_i, r_{i+1}, \ldots, r_j$ a la actividad d_l ofrece un beneficio $v(i, j, l)$. Deseamos encontrar la asignación de recursos a actividades que proporciona el máximo beneficio.

Se pide:

a) Formaliza el problema en términos de optimización,

b) plantea la ecuación recursiva de *programación dinámica* que calcula el mayor beneficio resultante de cualquier combinación válida de asignaciones, indicando cuál debe ser la llamada recursiva que se efectue a dicha ecuación para resolver el problema,

c) representa el grafo de dependencias entre las llamadas recursivas asociadas a la anterior ecuación para la instancia $R = 6$ y $D = 3$, y

d) diseña un algoritmo iterativo de *programación dinámica* lo más eficiente posible que recorra el anterior grafo y devuelva el beneficio máximo que puede obtenerse y las asignaciones que dan lugar a ese beneficio, indicando y justificando sus costes espacial y temporal.