Análisis de algoritmos y Estructuras de datos
21 de febrero de 2008 (Mañana)

Prueba individual

Nombre: **Grupo:**

Análisis de algoritmos

Dado un vector de enteros positivos a y un valor x, los dos siguientes algoritmos informan de si existen en el vector al menos dos números cuya suma es menor o igual que dicho valor:

```
def almenosdossuman_1(a,x):
  for i in xrange(len(a)):
    if a[i]<x:
      for j in xrange(i+1,len(a)):
        if a[i]+a[j]<=x:
          return True
  return False
```

```
def almenosdossuman_2(a,x):
  b=sorted(a)
  if b[0]+b[1]<=x:
    return True
  return False
```

a) 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. Si tuvieras que elegir uno de los dos algoritmos para resolver el problema, ¿con cuál te quedarías? Justifica tu respuesta.

b) ¿Se te ocurre algún método alternativo que garantizara un tiempo máximo menor que el de cualquiera de los dos algoritmos anteriores? Si es así, descríbelo brevemente y justifica su coste temporal.

Estructuras de datos (min-heap)

a) De los siguientes vectores, ¿cuáles podrían corresponderse con un min-heap y cuáles no? Justífica tu respuesta.

- $[1, 30, 7, 40, 50, 10]$
- $[1, 7, 9, 3, 5, 11]$
- $[1, 4, 6, 7, 5, 3, 2]$

b) Dado el vector $[20, 33, x, 100, 40, 39]$, indica en que rango de valores debería estar el elemento x para que el vector representara un min-heap.

c) El vector $a = [10, 22, 18, 30, 25, 40, 60, 50]$ representa un min-heap. Indica cuál sería su estado tras ejecutar la operación $a.extract\_min()$: dibuja el árbol antes y después de la operación.

Prueba colectiva

Grupo:
Asistentes:

Estructuras de datos y costes

Deseamos incorporar a la estructura de datos grafo (representado mediante conjuntos de adyacencia con inversa) una función adicional para combinar grafos, *difsim*(G_1, G_2), que reciba dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, con $V_1 = V_2$, y devuelva un nuevo grafo $G = (V, E)$, donde $V = V_1 = V_2$ y el conjunto de aristas E debe contener las aristas que están en E_1 pero no en E_2 y las que están en E_2 pero no en E_1.

Debes describir brevemente (y con claridad) como implementarías esta nueva operación, teniendo en cuenta que puedes utilizar los métodos ya disponibles en la interfaz de la estructura grafo, y de forma que sea lo más eficiente posible. Debes indicar, justificándolos, los costes temporal y espacial asociados a la función.

Análisis de algoritmos y Estructuras de datos
21 de febrero de 2008 (Tarde)

Prueba individual

Nombre: **Grupo:**

Análisis de algoritmos

Dado un vector de enteros positivos a y un valor x, los dos siguientes algoritmos calculan cuál es el máximo número de elementos del vector que, sumados, dan un resultado menor o igual que x:

```
def cuantossuman_1(a,x):
    suma=0
    b=a[:]
    for k in xrange(len(a)):
        suma+=min(b)
        if suma>x:
            return k
        else:
            b.remove(min(b))
    return len(a)
```

```
def cuantossuman_2(a,x):
    suma=0
    b=sorted(a)
    for k in xrange(len(a)):
        suma+=b[k]
        if suma>x:
            return k
    return len(a)
```

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? Justifica tu respuesta.

Estructuras de datos (MFSET)

Dado el siguiente MFSET:

	0	1	2	3	4	5	6	7	8	9
S	0	0	0	2	2	5	5	6	7	5

a) Dibuja cada uno de los árboles que representan los subconjuntos disjuntos que contiene S.

b) Indica sobre la siguiente tabla el contenido de la estructura después de realizar de forma consecutiva cada una de las operaciones que se indican a continuación, teniendo en cuenta que se emplean los heurísticos de *unión por rango* y *comprensión de caminos*. En la unión, en caso de que la altura de los dos árboles sea idéntica, el árbol del segundo parámetro será hijo del del primero. (Si lo prefieres, puedes rellenar únicamente los valores de las casillas que se modifican desde la anterior operación):

	0	1	2	3	4	5	6	7	8	9
S	0	0	0	2	2	5	5	6	7	5
$S.find(3)$										
$S.merge(9,1)$										
$S.find(4)$										

Prueba colectiva

Grupo:
Asistentes:

Estructuras de datos y costes

Deseamos incorporar a la estructura de datos diccionario una función adicional para combinar diccionarios, $unir(D_1, D_2)$, que reciba dos diccionarios D_1 y D_2 y devuelva un nuevo diccionario D que contenga todos los elementos de los dos diccionarios. En el caso en que hubiera elementos en D_1 y D_2 con la misma clave, sólo debería almacenarse en D el elemento (obviamente, una vez) si además de la clave coinciden sus valores en D_1 y D_2; si sólo coincidieran las claves, no se almacenaría ninguno de los dos elementos.

Debes describir brevemente (y con claridad) como implementarías esta nueva operación, teniendo en cuenta que puedes utilizar los métodos ya disponibles en la interfaz de la estructura, y de forma que sea lo más eficiente posible. Debes indicar, justificándolos, los costes temporal y espacial asociados a la función.

Divide y vencerás
12 de marzo de 2008 (Mañana)

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, 5, 8, 12, 9, 7, 20, 22, 6, 3, 17, 15, 14, 14], 7, 15)`.

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	5	8	12	9	7	20	22	6	3	17	15	14	14

2. Planteamiento y análisis de un problema

Dada f, una función creciente en un intervalo cerrado de valores enteros $[a, b]$ ($a < b$, $a, b \in \mathbb{Z}$) queremos averiguar si la función llega a valer 0 para algún punto incluido en dicho intervalo (si $f(y) = 0$, donde $y \in \mathbb{R}$) y, en tal caso, cuál sería el valor entero z ($z \in \mathbb{Z}$, $a \leq z < b$) anterior a dicho punto ($z \leq y < z + 1$).

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 estrategia planteada, dando una breve justificación del mismo.

Divide y vencerás
12 de marzo de 2008 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema

Dada f, una función creciente en un intervalo cerrado de valores enteros $[a, b]$ ($a < b$, $a, b \in \mathbb{Z}$) queremos averiguar si la función llega a valer 0 para algún punto incluido en dicho intervalo (si $f(y) = 0$, donde $y \in \mathbb{R}$) y, en tal caso, cuál sería el valor entero z ($z \in \mathbb{Z}$, $a \leq z < b$) anterior a dicho punto ($z \leq y < z + 1$).

a) Diseñad un algoritmo recursivo *mediante la técnica de divide y vencerás* que devuelva el valor z (en el caso en que no existiera, deberá devolver un valor especial). 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.

Divide y vencerás
12 de marzo de 2008 (Tarde)

Prueba individual

Nombre: **Grupo:**

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

Dado el vector

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
a	22	20	14	13	9	8	7	6	5	8	12	14	15	17

escribe todas las llamadas recursivas que se generan a partir de la ejecución de *convex_min*(*a*) (la versión de la página 5-31). Para cada llamada debes señalar los valores de los parámetros i y k en la función recursiva. ¿Cúal es el valor devuelto por el algoritmo?

2. Planteamiento y análisis de un problema

Disponemos de un vector a que contiene valores enteros y en el que se cumple que todos los valores impares del vector son consecutivos y aparecen ordenados de menor a mayor y agrupados en una parte del vector, y lo mismo sucede con los valores pares, que se agrupan en la otra parte. Un ejemplo de vector sería el siguiente: $a = [7, 9, 11, 13, 15, 10, 12, 14, 16, 18, 20, 22, 24]$. Queremos obtener la suma de los valores pares y la de los valores impares del vector, sabiendo que en un rango consecutivo de n valores ordenados (pares o impares) entre i y j (a_i es el primer elemento del rango y a_j el último) la suma de todos los elementos del rango es $(a_i + a_j) * (j - i + 1)/2$.

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
12 de marzo de 2008 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema

Disponemos de un vector a que contiene valores enteros y en el que se cumple que todos los valores impares del vector son consecutivos y aparecen ordenados de menor a mayor y agrupados en una parte del vector, y lo mismo sucede con los valores pares, que se agrupan en la otra parte. Un ejemplo de vector sería el siguiente: $a = [7, 9, 11, 13, 15, 10, 12, 14, 16, 18, 20, 22, 24]$. Queremos obtener la suma de los valores pares y la de los valores impares del vector, sabiendo que en un rango consecutivo de n valores ordenados (pares o impares) entre i y j (a_i es el primer elemento del rango y a_j el último) la suma de todos los elementos del rango es $(a_i + a_j) * (j - i + 1)/2$.

a) Diseñad un algoritmo recursivo *mediante la técnica de divide y vencerás* que devuelva la suma de los valores impares y la de los pares (podéis asumir que los valores impares aparecen en el vector antes que los pares y que como mínimo el vector contendra un valor par y uno impar). El algoritmo desarrollado debe ser lo más eficiente posible, tanto desde el punto de vista de la complejidad temporal como de la espacial.

   Tened en cuenta que la función recursiva que desarrolléis no tiene porque calcular directamente las sumas de los grupos, sino que puede devolver algún resultado que permita, una vez acabado el cálculo recursivo (y, por tanto, fuera de la recursión), obtener los valores pedidos. En cualquier caso, la estrategia principal deberá ser la de divide y vencerás y deberéis incluir todas las instrucciones necesarias para calcular el resultado pedido.

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) Adaptad vuestro algoritmo para que (si no lo hace ya) contemple la posibilidad de que en el vector pudieran estar al principio tanto los pares como los impares y que, además, el vector pudiera no contener valores de un tipo (en tal caso, la suma de ese tipo debería ser 0). En ningún caso deberán empeorar los costes del algoritmo con respecto al anterior.

Búsqueda con retroceso
3 de abril de 2008 (Mañana)

Prueba individual

Nombre: **Grupo:**

Traza problema de la suma del subconjunto **50 puntos**

En el problema de la suma del subconjunto disponemos de N objetos con pesos $w_1, w_2, \ldots, w_N$ y de una mochila con capacidad para soportar una carga W. Deseamos cargar la mochila con una selección arbitraria de objetos cuyo peso sea *exactamente* W. La función `subset_sum` (*la versión refinada del algoritmo de la página 6-23*) permite encontrar una solución factible para el problema aplicando la técnica de búsqueda con retroceso:

```
def subset_sum(w, W):
  w = OffsetArray(sorted(w))
  sum_w = OffsetArray([0]* (len(w)+1))
  sum_w[len(w)] = w[len(w)]
  for i in xrange(len(w)-1, 0, -1): sum_w[i] = sum_w[i+1] + w[i]
  x = OffsetArray([0] * len(w))

  def backtracking(i, W):
    if W == 0:
      for j in xrange(i, len(w)+1): x[j] = 0
      return x
    elif i <= len(w):
      for x[i] in 1, 0:
        if W-x[i]*w[i] >= 0 and sum_w[i+1] >= W-x[i]*w[i]:
          found = backtracking(i+1, W-x[i]*w[i])
          if found != None: return found
        else:
          return None
    return None

  return backtracking(1, W)
```

Realiza una traza de la ejecución del método para una mochila con capacidad para $W = 11$ y $N = 6$ objetos de pesos $w = [7, 1, 9, 2, 6, 4]$.

Debes dibujar el árbol de estados con tan sólo aquellos estados *que se exploran* durante el proceso de búsqueda y señalando el valor del peso total que queda por rellenar en la mochila al lado de cada estado. Incluye los estados que se estudian pero se descartan por no superar la condición de ser prometedores y márcalos de forma especial. Indica también claramente cuál es el resultado devuelto por el algoritmo. ¿Qué objetos son los seleccionados?

Búsqueda con retroceso
3 de abril de 2008 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

El problema del coloreado de un grafo con un máximo de k colores consiste en asignar un color a cada uno de los vértices de un grafo no dirigido de modo tal que dos vértices adyacentes no tengan asignado el mismo color y no se utilicen más de k colores.

Dado un grafo no dirigido $G = (V, E)$ y un número máximo de colores k, debes diseñar un algoritmo de *búsqueda con retroceso* que devuelva como resultado el número de color (entre 1 y k) asignado a cada uno de los vértices para que se cumplan las condiciones del 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 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. En el problema original del coloreado de un grafo no se proporciona un número máximo de colores k, se propone la búsqueda del coloreado que emplea *el menor número de colores*. ¿Serías capaz de adaptar tu algoritmo para que resolviera dicho problema?

Búsqueda con retroceso
3 de abril de 2008 (Tarde)

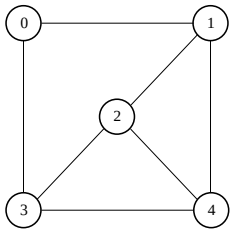
Prueba individual

Nombre: **Grupo:**

Traza problema de la existencia de un ciclo hamiltoniano **50 puntos**

En el problema de la existencia de un ciclo hamiltoniano, dado un grafo $G = (V, E)$, deseamos encontrar un camino que parta de un vértice, termine en el mismo vértice y visite todos los vértices del grafo una sola vez (excepto en el caso del vértice de partida, que deberá coincidir con el de llegada). El algoritmo `hamiltonian_cycle` (*la versión refinada de la página 6-32*) resuelve el problema aplicando la técnica de búsqueda con retroceso.

Realiza una traza de la ejecución del método para la llamada `hamiltonian_cycle(G)`, donde G es el grafo que se muestra a continuación. Para la traza, considera que el vértice de partida seleccionado aleatoriamente es el 0 y que la función $G.succ(v), \forall v \in G.V$, devuelve *siempre* una lista con los vértices sucesores de v ordenados de menor a mayor valor númerico).



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

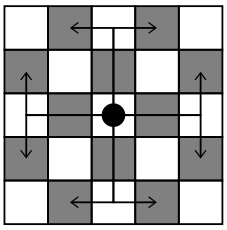
Búsqueda con retroceso
3 de abril de 2008 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

El caballo de ajedrez puede, desde una casilla del tablero, desplazarse a una cualquiera de las ocho casillas señaladas con una flecha en la siguiente figura:



Debes diseñar un algoritmo de *búsqueda con retroceso* que, dada una casilla de partida en la que se sitúa un caballo, encuentre un recorrido en un tablero de ajedrez de $n \times n$ con el caballo de modo que visite todas las casillas, pero que ninguna la visite más de una vez.

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. En el problema original del recorrido del caballo no se facilita ninguna casilla de partida (en algunos casos salir de una casilla concreta podría dar lugar a que no hubiera solución). ¿Serías capaz de adaptar tu algoritmo para que resolviera dicho problema?

Algoritmos voraces
21 de abril de 2008 (Mañana)

Prueba individual

Nombre: **Grupo:**

Selección y adaptación de un algoritmo voraz **35 puntos**

Un organismo regional encargado del control de epidemias de origen animal tiene una estimación del número de horas que tardaría en propagarse un brote de gripe aviaria entre cada par de granjas de la región dedicadas a la cria y venta de pollos.

Por ejemplo, si en la región hay 5 granjas (A, B, C, D y E), se sabe que desde la granja A se puede propagar la gripe a la granja B en 48 horas, a D en 30 horas y a E en 56 horas; desde B se puede propagar la enfermedad a A en 60 horas, a C en 72 horas y a D en sólo 24 horas; desde C no se puede propagar a ninguna otra granja; desde D sólo se puede propagar a C en 10 horas y desde E sólo a D en 15 horas. En los casos no indicados, no hay posibilidad de extensión de la epidemia.

A partir de los tiempos de propagación entre pares de granjas, dadas también las granjas en las que se detecta el inicio de un brote vírico (que podría surgir *simultaneamente* en más de una a la vez) y proporcionado un tiempo máximo estimado de respuesta, los responsables del organismo desean disponer de un programa informático capaz de determinar qué granjas se han contagiado.

Siguiendo con el anterior ejemplo, si la epidemía se ha iniciado en B y el tiempo de respuesta es de 48 horas, el programa devolvería $\{B, D, C\}$: obviamente, B ya está infectado, D se contagiaría directamente de B a las 24 horas (≤ 48) y C se contagiaría a través de D en $24 + 10 = 34$ horas (≤ 48). En otro caso, con A y E como focos simultaneos de la infección y un tiempo de respuesta de 30 horas, la solución devuelta sería $\{A, E, D, C\}$.

Se pide:

a) Indica qué estructura de datos elegirías para representar las granjas y tiempos de propagación de la gripe entre granjas. Representa gráficamente el contenido de la estructura de datos escogida para los datos del ejemplo del segundo párrafo.

b) Alguno de los algoritmos que hemos estudiado en el tema permite abordar problemas similares a este. Es posible que haya que adaptarlo para que resuelva aspectos concretos de este problema. Indica qué algoritmo emplearías y qué modificaciones concretas le aplicarías para resolver el problema (no es necesario que las codifiques, pero si que las describas con precisión).

c) ¿Qué costes temporales presenta la versión del algoritmo que has descrito previamente en el mejor y el peor de los casos?

Algoritmos voraces
21 de abril de 2008 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Análisis de la solución voraz de un problema — 65 puntos

El problema de la *cobertura del conjunto* se define con un conjunto A y una familia $F = \{F_1, F_2, ..., F_N\}$ de N subconjuntos de A cuya unión coincide con A. El objetivo es seleccionar el menor número de subconjuntos de F tal que su unión contenga a todos los elementos de A. Un ejemplo ayudará a entender el problema. Sea $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y sea $F = \{F_1, F_2, F_3, F_4\}$, donde $F_1 = \{1, 2, 3, 4, 5\}$, $F_2 = \{4, 5, 6, 9\}$, $F_3 = \{1, 2, 7, 8, 9\}$ y $F_4 = \{6, 7, 8\}$. Nótese que A es la unión de todos los conjuntos de F. La unión $F_1 \cup F_2 \cup F_3$ es igual a A, así que $\{F_1, F_2, F_3\}$ es una cobertura del conjunto y tiene talla 3. Como no hay otra cobertura posible con un número menor de elementos de F, se trata de una cobertura óptima. Me han propuesto el siguiente algoritmo voraz para resolver el problema:

```
 1 def greedy_set_cover(A, F):
 2   AA = A.copy()  # A.copy() crea una copia del conjunto.
 3   FF = F.copy()
 4   C = set()
 5   while len(AA) > 0:
 6     # Selecciono el subconjunto Fi de FF que añade más elementos de AA.
 7     inAA = -1
 8     for Fi in FF:
 9       num = len(Fi.intersection(AA))
10       if num > inAA:
11         inAA = num
12         S = Fi
13     FF.remove(S)
14     C.add(S)
15     # Actualizo AA, que es el conjunto de elementos de A que aún no he añadido.
16     AA.difference_update(S)
17   return C
```

He aquí un ejemplo de uso de esa función:

```
1 A = set(xrange(1,10))
2 F = set([frozenset([1,2,3,4,5]), frozenset([4,5,6,9]), frozenset([1,2,7,8,9]), frozenset([6,7,8])])
3 print greedy_set_cover(A, F)
```

Responde *razonadamente* a las siguientes preguntas:

a) ¿Encuentra siempre una solución factible?

b) ¿Encuentra siempre una solución óptima?

c) ¿Qué coste temporal presenta en el mejor y el peor caso?

d) ¿Qué coste espacial presenta?

Algoritmos voraces
21 de abril de 2008 (Tarde)

Prueba individual

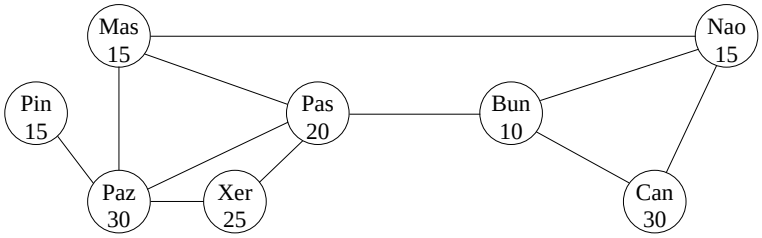
Nombre: **Grupo:**

Dijkstra — 35 puntos

Tenemos un mapa de carreteras con ciudades que podemos modelar con un grafo. Un coche fúnebre debe transportar un cadáver desde una ciudad s a otra ciudad t. La legislación obliga a pagar un impuesto funerario en cada *ciudad* por la que pase el coche (incluyendo las ciudades inicial y final). Nos gustaría saber el mínimo precio con el que podemos ir de s a t, así que hemos pensado en aplicar el algoritmo de Dijkstra (los impuestos que hemos de pagar son siempre cantidades no negativas), pero nos hemos encontrado con una seria dificultad: ¡el impuesto grava el tránsito por la ciudad, no por la carretera! ¿Qué hemos de hacer para poder utilizar Dijkstra en este problema?

Cuando lo tengas claro, cuéntamelo. Puede que tengas que modificar el algoritmo de Dijkstra y puede que no. Puede que tengas que modificar el grafo y puede que no. Hagas lo que hagas, cuéntamelo.

A continuación haz una traza de Dijkstra (posiblemente modificado) sobre el siguiente mapa (posiblemente modificado) al tratar de ir de Mas a Bun (en cada ciudad hemos puesto el nombre de la ciudad y el impuesto funerario asociado):



Puedes presentar la traza gráficamente mostrando la evolución del algoritmo mediante figuras (como hicimos en clase al presentar el ejemplo de Mallorca) o bien mostrando el contenido de las principales estructuras de datos del algoritmo. No es necesario que detalles cada paso de la traza, pero sí, como mínimo, debes indicar el contenido final del diccionario D y la solución devuelta por el algoritmo.

Algoritmos voraces
21 de abril de 2008 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Análisis de la solución voraz de un problema — 65 puntos

He de matricularme de K créditos para acabar mis estudios. La universidad me ofrece N asignaturas (numeradas entre 1 y N), cada una con una carga expresada en créditos y que denotamos con $c_1, c_2, \ldots, c_N$, respectivamente. Consultando las actas del pasado curso he confeccionado una tabla con la tasa de aprobados en cada asignatura. Con $a_1, a_2, \ldots, a_N$ (números entre 0 y 1) denoto la tasa de aprobados en cada asignatura. Ese valor constituye para mí una estimación de la probabilidad con la que aprobaré la asignatura en caso de cursarla. Una matrícula es un conjunto de asignaturas $\{x_1, x_2, \ldots, x_m\}$ (es decir, $x_i \in [1..N]$ y $x_i \neq x_j, 1 \leq i \neq j \leq m$) y entiendo por matrícula óptima la que hace máximo el valor de $\sum_{1 \leq i \leq m} a_{x_i} c_{x_i}$. Trabajaremos bajo dos supuestos:

Uno Que mi pereza llega al extremo de no querer cursar ni un crédito más de los K que necesito para completar la carrera.

Dos Que mi deseo de aprender llega al punto de aceptar cursar algunos créditos de más, pero sólo si son el excedente al que me obliga la asignatura con la que completo los K créditos que he de cursar.

Me propongo escoger las asignaturas siguiendo uno de estos dos algoritmos:

1. Me voy matriculando de asignaturas en orden de mayor a menor valor de a_{x_i}.
2. Me voy matriculando de asignaturas en orden de mayor a menor valor del producto $a_{x_i} c_{x_i}$.

Responde *razonadamente* a las siguientes cuestiones:

a) ¿Obtendremos una matrícula óptima en el supuesto **Uno** con el algoritmo 1?

b) ¿Obtendremos una matrícula óptima en el supuesto **Uno** con el algoritmo 2?

c) ¿Obtendremos una matrícula óptima en el supuesto **Dos** con el algoritmo 1?

d) ¿Obtendremos una matrícula óptima en el supuesto **Dos** con el algoritmo 2?

e) ¿Qué coste temporal presentan los algoritmos 1 y 2 en el supuesto **Uno**? (Expresa una cota tan ajustada como te sea posible.)

f) ¿Qué coste temporal presentan los algoritmos 1 y 2 en el supuesto **Dos**? (Expresa una cota tan ajustada como te sea posible.)

Programación dinámica
30 de abril de 2008 (Mañana)

Prueba individual

Nombre: **Grupo:**

Problema de la mochila discreta **50 puntos**

Dada una mochila con capacidad para cargar $W = 4$ unidades de peso y $N = 6$ objetos que podemos cargar en ella, con pesos $w = [3, 4, 2, 1, 3, 1]$ y valores $v = [20, 30, 15, 9, 25, 5]$, 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):

(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)
(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)

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 se encuentra? ¿Qué resultado devuelve el algoritmo? ¿Qué significa este último resultado?

Programación dinámica
30 de abril de 2008 (Mañana)

Prueba colectiva

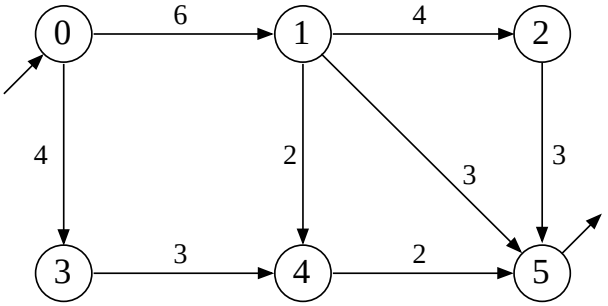
Grupo:
Asistentes:

Recuperación voraces **+25 puntos (añadir a la puntuación de voraces)**

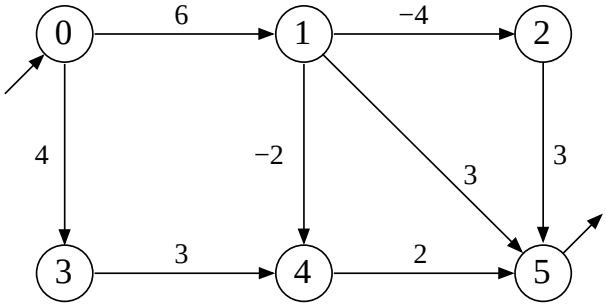
Conoceis dos algoritmos capaces de calcular el camino más corto entre dos vértices en un grafo dirigido ponderado: el algoritmo de Dijkstra (voraz) y el del camino más corto en un grafo acíclico (de programación dinámica).

Para cada uno de los siguientes grafos, en los que se quiere obtener el camino más corto entre el vértice 0 y el vértice 5, escribe debajo **"Ninguno", "Dijkstra", "PD", "Los dos"** para indicar qué algoritmo o algoritmos podrían aplicarse para resolver el problema. En el caso en que se pudieran emplear los dos, señala cuál consideras más adecuado:

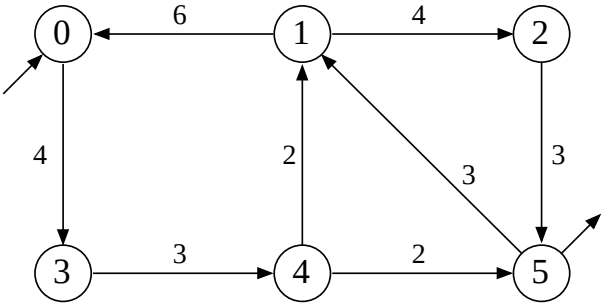
a)



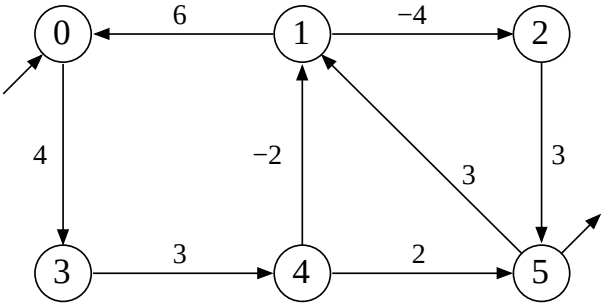
b)



c)



d)



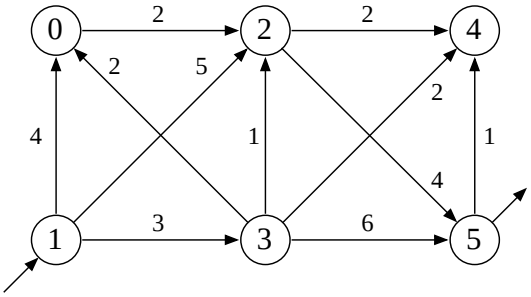
Programación dinámica
30 de abril de 2008 (Tarde)

Prueba individual

Nombre: **Grupo:**

Problema del camino más corto en un grafo acíclico **50 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:



Para ello, se pide:

a) Dibuja el grafo ordenado topológicamente.

b) Haz una traza del algoritmo `dag_shortest_path` (pág 7-34), el que permite recuperar el camino óptimo. Para ello puedes, bien utilizar como base el grafo obtenido en el apartado anterior de forma similar a cómo se hacía en el apartado (g) de la figura 7.23 (pág 7-35), bien rellenar los diccionarios D y B. *No* se pide una traza iteración a iteración, basta con que se muestre el contenido final de las dos estructuras citadas.

c) ¿Cuál es el valor de la solución óptima? ¿En qué estado se encuentra? ¿Qué resultado devuelve el algoritmo? ¿Qué significa este último resultado?

Programación dinámica
30 de abril de 2008 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Recuperación voraces **+25 puntos (añadir a la puntuación de voraces)**

La representación formal del problema de la mochila *sin fraccionamiento* sabemos que es la siguiente:

- Conjunto de soluciones factibles:

$$X = \left\{ (x_1, x_2, \ldots, x_N) \in \{0,1\}^N \middle| \sum_{1 \leq i \leq N} x_i w_i \leq W \right\}.$$

- Optimización y función objetivo:

$$\arg \max_{(x_1, x_2, \ldots, x_N) \in X} \sum_{1 \leq i \leq N} x_i v_i.$$

¿Eres capaz de indicar qué variaciones se han introducido en el *enunciado original* del problema para cada uno de los siguientes casos en los que la parte de la formalización que se modifica aparece en un recuadro?:

a) Modificación en la definición del conjunto de soluciones factibles:

$$X = \left\{ (x_1, x_2, \ldots, x_N) \in \{0,1\}^N \middle| \boxed{W' \leq} \sum_{1 \leq i \leq N} x_i w_i \leq W \right\}.$$

b) Modificación en la definición de la función objetivo:

$$\arg \max_{(x_1, x_2, \ldots, x_N) \in X} \sum_{1 \leq i \leq N} \boxed{x_i w_i}.$$

c) Modificación en la definición del conjunto de soluciones factibles:

$$X = \left\{ (x_1, x_2, \ldots, x_N) \in \{0,1\}^N \middle| \sum_{1 \leq i \leq N} x_i w_i \leq W; \boxed{N/5 \leq \sum_{1 \leq i \leq N} x_i \leq N/2} \right\}.$$

Programación dinámica
13 de mayo de 2008 (Mañana)

Prueba individual

Nombre: **Grupo:**

Problema **50 puntos**

Una importante empresa de inversiones ha diseñado una prueba para los aspirantes a trabajar en ella: les asigna una cantidad de dinero y, durante T meses, deben hacer un total de M inversiones de dicha cantidad con el objetivo de obtener el máximo beneficio. Para ello, pueden invertir el dinero a plazo fijo en N diferentes productos financieros. Cada producto i ($1 \leq i \leq N$) tiene asociado el plazo de recuperación $d(i)$ y el beneficio $b(i)$. El procedimiento de inversión consiste en seleccionar un producto i, invertir el capital completo en él y, transcurrido el tiempo $d(i)$, recoger el beneficio y repetir el proceso. No hay inconveniente en invertir en un mismo producto más de una vez. Durante todo el periodo T, el capital debe estar invertido en algún producto y, una vez finalizado el periodo, hay que haber recuperado el capital inicial.

Deseas saber en qué productos realizarás las inversiones para obtener el máximo beneficio posible.

La siguiente ecuación recursiva te devuelve el valor del máximo beneficio que puedes obtener al hacer m inversiones durante t meses:

$$B(t,m) = \begin{cases} 0, & \text{si } t = 0 \text{ y } m = 0; \\ -\infty, & \text{si } (t > 0 \text{ y } m = 0) \text{ o } (t = 0 \text{ y } m > 0); \\ \max\limits_{\substack{1 \leq i \leq N: \\ d(i) \leq t}} B(t - d(i), m - 1) + b(i), & \text{si } t > 0 \text{ y } m > 0. \end{cases}$$

donde la llamada $B(T, M)$ proporciona el máximo beneficio para el tiempo total con todas las inversiones que hay que realizar. 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 $T = 6$, $M = 3$ y $N = 4$ productos financieros de duraciones $d = [1, 3, 5, 2]$ y beneficios $b = [10, 33, 57, 19]$.

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 2008 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

Sea el anterior problema:

****

Una importante empresa de inversiones ha diseñado una prueba para los aspirantes a trabajar en ella: les asigna una cantidad de dinero y, durante T meses, deben hacer un total de M inversiones de dicha cantidad con el objetivo de obtener el máximo beneficio. Para ello, pueden invertir el dinero a plazo fijo en N diferentes productos financieros. Cada producto i $(1 \leq i \leq N)$ tiene asociado el plazo de recuperación $d(i)$ y el beneficio $b(i)$. El procedimiento de inversión consiste en seleccionar un producto i, invertir el capital completo en él y, transcurrido el tiempo $d(i)$, recoger el beneficio y repetir el proceso. No hay inconveniente en invertir en un mismo producto más de una vez. Durante todo el periodo T, el capital debe estar invertido en algún producto y, una vez finalizado el periodo, hay que haber recuperado el capital inicial.

Deseas saber en qué productos realizarás las inversiones para obtener el máximo beneficio posible.

La siguiente ecuación recursiva te devuelve el valor del máximo beneficio que puedes obtener al hacer m inversiones durante t meses:

$$B(t,m) = \begin{cases} 0, & \text{si } t = 0 \text{ y } m = 0; \\ -\infty, & \text{si } (t > 0 \text{ y } m = 0) \text{ o } (t = 0 \text{ y } m > 0); \\ \max\limits_{\substack{1 \leq i \leq N: \\ d(i) \leq t}} B(t - d(i), m - 1) + b(i), & \text{si } t > 0 \text{ y } m > 0. \end{cases}$$

donde la llamada $B(T, M)$ proporciona el máximo beneficio para el tiempo total con todas las inversiones que hay que realizar.

****

Sobre el problema original se pide:

a) Escribid el algoritmo iterativo *con reducción de la complejidad espacial* que permite obtener el valor del máximo beneficio de la inversiones.

A continuación, se proponen las siguientes modificaciones en el enunciado del problema de las inversiones:

1. No es obligatorio hacer exactamente M inversiones.
2. No es obligatorio mantener el dinero invertido durante todos los meses, si conviene puede estar algún mes disponible.
3. El beneficio que se obtiene con un producto depende del mes en que se haya invertido, es decir, ahora cada producto i tendrá asociados T valores, $b(i,t)$, $(1 \leq t < T)$.

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 combinación válida de inversiones, 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 $T = 6$ meses y $N = 4$ productos financieros de duraciones $d = [1, 3, 5, 2]$.

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 indicado en el apartado anterior si hubiera que diseñar un algoritmo iterativo que sólo encontrara *el valor* del máximo beneficio? Si es así, indicad cuál sería dicho coste. Justificad vuestra respuesta.

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

Programación dinámica
13 de mayo de 2008 (Tarde)

Prueba individual

Nombre: **Grupo:**

Problema **50 puntos**

Se dispone de una cierta cantidad de dinero que se quiere invertir en un periodo de T meses. Al comienzo de cada mes t ($0 \leq t < T$) se puede elegir entre invertir todo el dinero en bonos, en cuyo caso el dinero invertido no se recupera hasta d_b meses después y produce un beneficio $b_b(t)$ que depende del mes de la inversión, o almacenar el dinero en la hucha, en cuyo caso estará disponible el mes siguiente pero sin producir beneficio alguno. Los beneficios que se van obteniendo no se vuelven a invertir, sino que se destinan directamente a una ONG, por lo que la cantidad de dinero disponible para cada inversión siempre es la misma. Una vez finalizado el periodo, el dinero tiene que estar nuevamente disponible.

Deseas saber cómo tienes que hacer las inversiones para obtener el máximo beneficio posible.

La siguiente ecuación recursiva te devuelve el valor del máximo beneficio que puedes obtener de tus inversiones durante t meses:

$$B(t) = \begin{cases} 0, & \text{si } t = 0; \\ B(t-1), & \text{si } 0 < t < d_b; \\ \text{máx}(B(t-1), B(t-d_b) + b_b(t-d_b)), & \text{si } t \geq d_b. \end{cases}$$

donde la llamada $B(T)$ proporciona el máximo beneficio para el tiempo total disponible. 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 $T = 8$, $d_b = 3$ y beneficios $b_b = [10, 20, 30, 25, 40, 10, 13, 35]$.

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 2008 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema **100 puntos**

Sea el anterior problema:

****

Se dispone de una cierta cantidad de dinero que se quiere invertir en un periodo de T meses. Al comienzo de cada mes t ($0 \leq t < T$) se puede elegir entre invertir todo el dinero en bonos, en cuyo caso el dinero invertido no se recupera hasta d_b meses después y produce un beneficio $b_b(t)$ que depende del mes de la inversión, o almacenar el dinero en la hucha, en cuyo caso estará disponible el mes siguiente pero sin producir beneficio alguno. Los beneficios que se van obteniendo no se vuelven a invertir, sino que se destinan directamente a una ONG, por lo que la cantidad de dinero disponible para cada inversión siempre es la misma. Una vez finalizado el periodo, el dinero tiene que estar nuevamente disponible.

Deseas saber cómo tienes que hacer las inversiones para obtener el máximo beneficio posible.

La siguiente ecuación recursiva te devuelve el valor del máximo beneficio que puedes obtener de tus inversiones durante t meses:

$$B(t) = \begin{cases} 0, & \text{si } t = 0; \\ B(t-1), & \text{si } 0 < t < d_b; \\ \max(B(t-1), B(t-d_b) + b_b(t-d_b)), & \text{si } t \geq d_b. \end{cases}$$

donde la llamada $B(T)$ proporciona el máximo beneficio para el tiempo total disponible.

****

Sobre el problema original se pide:

a) Escribid el algoritmo iterativo que proporciona *la secuencia de decisiones óptima, concretamente, debe proporcionar una lista de los meses en que se debe invertir en bonos* para obtener el máximo beneficio.

A continuación, se propone la siguiente modificación en el enunciado del problema:

1. Adicionalmente a invertir en bonos o dejarlo en la hucha, los meses en que tengamos el capital disponible también podemos invertirlo en *comprar acciones*. En el caso de las acciones, no tienen una fecha de recuperación fija, sino que pueden venderse en cualquier mes posterior al de compra. Asimismo, disponemos de una estimación del beneficio $b_a(t', t)$ que obtendremos si vendemos en el mes t unas acciones compradas en el mes t'.

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 combinación válida de inversiones, indicando cuál debe ser la llamada inicial que se efectúe a dicha función para obtener el valor óptimo.

b) Representa el grafo de dependencias asociado a la ecuación recursiva para $T = 8$ y $d_b = 3$.

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 indicado en el apartado anterior si hubiera que diseñar un algoritmo iterativo que sólo encontrara *el valor* del máximo beneficio? Si es así, indicad cuál sería dicho coste. Justificad vuestra respuesta.

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

Primer control parcial de Esquemas Algorítmicos (IG24)

5 de abril de 2008

1. Divide y vencerás — 5 puntos

El problema de la pirámide maya consiste en determinar la altura de una pirámide a partir del pérfil de un fragmento de la misma que incluye su cima. Para ello, en un vector a disponemos de los datos de n mediciones con la altura de los distintos escalones, de izquierda a derecha, incluidos en el fragmento. Los valores de altura están ordenados de menor a mayor hasta la cima y de mayor a menor a partir de esta. Como la anchura de los escalones es variable, y las mediciones se han hecho tomando un ancho fijo que garantice incluir hasta el escalón más estrecho, el vector podría contener en varias posiciones seguidas un mismo valor, que se correspondería con el de un mismo escalón. Veamos un ejemplo de vector de alturas: $a = [3, 3, 5, 5, 5, 5, 8, 8, 12, 12, 12, 9, 7, 7]$.

Disponemos asimismo de un vector auxiliar i_f que contiene, para cada posición/medición del vector a, las posiciones en que comienza y acaba el escalón correspondiente. El contenido del vector auxiliar asociado al anterior ejemplo sería $i\_f = [(0,1), (0,1), (2,5), (2,5), (2,5), (2,5), (6,7), (6,7), (8,10), (8,10), (8,10), (11,11), (12,13), (12,13)]$, donde, por ejemplo, la posición 8 (y la 9 y la 10) contiene el valor $i\_f[8] = (8, 10)$, que indica que el escalón comienza en la posición 8 y acaba en la 10. Se pide:

a) Diseña un algoritmo recursivo mediante la técnica de divide y vencerás que devuelva *la altura de la pirámide* representada en a. El algoritmo debe ser lo más eficiente posible.

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

e) En caso de no disponer del vector auxiliar i_f, ¿seguiría siendo válido el algoritmo que has desarrollado? Si no es así, ¿podrías describir que modificaciones introducirías en tu algoritmo para seguir resolviendo el problema de la forma más eficiente posible y cuál sería el coste temporal que tendría asociado?

2. Búsqueda con retroceso — 5 puntos

Tenemos una lista compuesta por n números enteros positivos, la mitad de los cuales son pares y la otra mitad impares (si n es impar, habrá un elemento más en uno de los dos tipos), dispuestos en cualquier orden.

Dados un valor de umbral mínimo d y un valor de umbral máximo D, queremos encontrar una ordenación de los números de la lista que cumpla que los pares y los impares se alternen y que la diferencia (en valor absoluto) entre cada dos números consecutivos de la ordenación sea mayor que d y menor que D.

Por ejemplo, si la lista fuese $L = [2, 5, 19, 29, 14, 10, 25, 22, 13, 31, 26, 30, 36]$ y los umbrales $d = 4$ y $D = 12$, una posible solución podría ser $[2, 13, 22, 29, 36, 31, 26, 19, 30, 25, 14, 5, 10]$. Queremos encontrar un algoritmo de *búsqueda con retroceso* que resuelva el problema de encontrar una ordenación 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).

e) Realiza una traza de ejecución de tu algoritmo (de la versión que prefieras, indica cuál) para el ejemplo $d = 3$, $D = 8$ y $L = [10, 12, 5, 15]$. Debes representar el árbol de llamadas efectuadas, incluyendo solamente aquellos estados que se generan durante el proceso de búsqueda. Marca de forma especial los estados que se estudian pero se descartan por no superar la condición de ser prometedores. Indica cuál es el resultado devuelto por el algoritmo.

Segundo control parcial de Esquemas Algorítmicos (IG24)

17 de mayo de 2008

1. Algoritmos voraces — 3.25 puntos

Alquilamos un coche para hacer un viaje por Europa por una ruta previamente fijada de K kilómetros. En nuestro coche caben m litros de gasolina y tiene una autonomia (distancia que puede recorrer con el depósito lleno de gasolina) de n kilómetros (con $m'(< m)$ litros el coche puede recorrer una distancia proporcionalmente inferior: $\frac{m'}{m}n$ kilómetros).

A lo largo de la ruta hay G gasolineras, númeradas de 1 a G según su orden de aparición en la ruta, y para cada una de ellas conocemos tanto el punto kilométrico de nuestra ruta en el que se encuentran, g_i, como el precio al que tienen el litro de la gasolina que emplea nuestro vehículo, p_i, siendo i la gasolinera i-ésima de la ruta ($1 \leq i \leq G$). La distancia entre dos gasolineras consecutivas siempre será menor o igual que la autonomía del coche n.

Quisieramos poder calcular antes de salir de viaje, *mediante un algoritmo voraz*, en qué gasolineras debemos parar a repostar y cuanta cantidad de combustible pondremos en el depósito en cada una de las paradas para gastarnos el menor dinero posible en gasolina en el viaje.

Inicialmente tenemos el deposito vacio, pero no importa, ya que la primera gasolinera está justo a la salida de la agencia de alquiler ($g(1) = 0$). Al llegar al destino (K) deberemos dejar el coche en una sucursal de la agencia (volvemos en avión), por lo que nos interesa que el depósito esté lo más vacio posible.

Ejemplo simplificado: Ruta de $K = 2250$ kilómetros. Depósito de $m = 50$ litros, con el que se pueden recorrer hasta $n = 750$ kilómetros (15 kilómetros por litro). Hay $G = 8$ gasolineras en la ruta, en las posiciones kilométricas $g = [0, 150, 750, 1050, 1350, 1650, 1950, 2100]$ y con precios por litro $p = [1.25, 1.1, 1.12, 1.05, 1.3, 1.5, 1.6, 1.15]$. Una solución para nuestro problema sería [(1,50), (3,50), (5,40), (8,10)], con un coste de $50*1.25+50*1.12+40*1.3+10*1.15 = 182$ euros. (La solución óptima sería [(1,10), (2,50), (3,10), (4,50), (5,20), (8,10)], con un coste de $10*1.25+50*1.1+10*1.12+50*1.05+20*1.3+10*1.15 = 168.7$ euros.)

Se pide:

a) Describe con la mayor precisión (puedes ayudarte de diagramas si lo consideras conveniente) una estrategia voraz que proporcione la lista de gasolineras y litros que tenemos que repostar en cada una de ellas para gastarnos la menor cantidad de dinero posible en gasolina.

b) Indica cuál es el coste temporal de la estrategia que has diseñado, justificándolo adecuadamente.

c) ¿Es la estrategia que has diseñado la más eficiente posible? Si crees que puedes mejorar sus costes (por ejemplo, haciendo uso de determinadas estructuras de datos), indica cómo y señala cuál sería el nuevo coste temporal de la estrategia.

2. Programación dinámica — 4 puntos

Disponemos de una cisterna con una cantidad ilimitada de agua y deseamos llenar una cuba con exactamente M litros. Tenemos un juego de J jarras, y conocemos la capacidad en litros $l(j)$ de cada jarra j ($1 \leq j \leq J$) (todas las capacidades son cantidades enteras). Nuestro objetivo es llenar completamente la cuba de agua usando las jarras para llevar el agua desde la cisterna.

Concretamente, queremos realizar exactamente K viajes para llenar completamente la cuba, teniendo en cuenta que en un viaje sólo podemos trasladar una jarra de agua, que la jarra debe estar completamente llena y que toda el agua que se lleva se debe depositar en la cuba (no se puede desechar agua). Es posible utilizar una misma jarra en más de un viaje.

Cada viaje supone un esfuerzo que podemos cuantificar en función de la cantidad de agua que trasladamos como el cuadradado del peso de la cantidad de agua trasladada (es decir, si usamos la jarra j, el esfuerzo asociado sería $l(j)^2$). Si el esfuerzo total realizado es la suma de los esfuerzos de cada viaje, ¿cuál es la combinación de jarras que debemos utilizar en K viajes para llenar los M litros de la cuba de forma que hagamos el *mínimo* esfuerzo total?

1. Con las condiciones anteriormente expuestas:

   a) Formaliza el problema en términos de optimización.

   b) Plantea la ecuación recursiva asociada al hecho de llenar la cuba con agua haciendo K viajes de forma que estos supongan el menor esfuerzo posible. Indica cuál debe ser la llamada que se efectue a dicha función para obtener el valor óptimo.

   c) Representa el grafo de dependencias (sin extender) entre llamadas de la ecuación recursiva para el ejemplo $M = 10$, $K = 3$, $J = 4$ y $l = [1, 2, 4, 5]$, razonando 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 del esfuerzo mínimo).

   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 restringimos los viajes a un número determinado K, nos interesa simplemente la combinación de viajes que presenta el menor esfuerzo; ahora bien, hemos encontrado una forma más ajustada de medir el esfuerzo, y es que este depende, no sólo de los litros que transportemos en un viaje, sino del número de viajes que previamente hayamos realizado, con la siguiente formula: $l(j)^2 + k^2$, que representa el esfuerzo de transportar la jarra j habiendo realizado hasta el momento un total de k viajes. Resumiendo, el problema pasa a

ser cómo llenar la cuba haciendo cualquier número de viajes con jarras de manera que el esfuerzo total sea mínimo (aplicando la nueva fórmula de esfuerzo de un viaje que tiene en cuenta el número de viajes previos realizados).

¿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

Sean dos sucursales de una casa de suministro de tornillos que sirven a N tiendas, siendo $d(i)$ la demanda de tornillos de la tienda i ($1 \leq i \leq N$). El número de tornillos que tiene disponible cada sucursal es R_j ($j \in \{1, 2\}$). Se cumple que la demanda total de tornillos, $\sum_{1 \leq i \leq N} d(i)$, coincide con los tornillos totales disponibles, $R_1 + R_2$. Enviar los $d(i)$ tornillos que solicita la tienda i desde la sucursal j tiene un coste de $c(j, i)$.

Debes averiguar, empleando la técnica de programación dinámica, qué sucursal deber realizar cada envío de forma que se cubra toda la demanda con el menor coste posible para la casa de suministros. Como ayuda, la siguiente ecuación recursiva de programación dinámica permite resolver el problema de obtener el valor del suministro más barato:

$$C(i, r_1, r_2) = \begin{cases} 0, & \text{si } i = 0; \\ +\infty, & \text{si } i > 0 \text{ y } r_1 < d(i) \text{ y } r_2 < d(i); \\ C(i-1, r_1, r_2 - d(i)) + c(2, i), & \text{si } i > 0 \text{ y } r_1 < d(i) \text{ y } r_2 \geq d(i); \\ C(i-1, r_1 - d(i), r_2) + c(1, i), & \text{si } i > 0 \text{ y } r_1 \geq d(i) \text{ y } r_2 < d(i); \\ \text{mín}(C(i-1, r_1 - d(i), r_2) + c(1, i), C(i-1, r_1, r_2 - d(i)) + c(2, i)), & \text{en otro caso.} \end{cases}$$

La llamada $C(N, R_1, R_2)$ proporciona dicho valor.
Se pide:

a) Representa el grafo de dependencias (sin extender) para el problema en el que $N = 5$, $d = [3, 4, 2, 1, 6]$, $R_1 = 7$, $R_2 = 9$, $c[1] = [2, 3, 4, 3, 5]$ y $c[2] = [3, 4, 1, 2, 6]$ (no te agobies si no te sale muy ordenado). Indica qué representa el estado $(3, 6, 3)$ y cómo interpretarías el significado de un arco entre los estados $(2, 6, 1)$ y $(3, 6, 3)$ dentro del grafo.

b) Diseña un algoritmo iterativo que devuelva, para cada tienda, *cuál es la sucursal que se encarga de su envío* para garantizar que se realiza el menor gasto.

c) Indica cuál es el coste temporal y espacial del algoritmo diseñado justificando brevemente tu respuesta.

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

Examen convocatoria ordinaria de Esquemas Algorítmicos (IG24)

13 de junio de 2008

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

Tenemos una foto con un punto de luz y la iluminación que produce. Dada una matriz de $n \times m$ que representa la digitalización de los distintos tonos de gris de la foto, debes localizar en qué posición de la matriz se encuentra el punto de luz. Los valores almacenados en la matriz pueden variar entre el 0 (blanco) y el 65535 (negro). Debes tener en cuenta que la matriz contiene una gradación de forma concéntrica de tonos (valores enteros crecientes) desde el punto central y en círculo hasta los puntos más alejados. Veamos un ejemplo de matriz de 10×15:

```
luz =
[[ 25, 25, 25, 25, 25, 25, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 20, 20, 20, 20, 20, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 15, 15, 15, 15, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 15, 10, 10, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 15, 10,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 15, 10, 10, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 15, 15, 15, 15, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 20, 20, 20, 20, 20, 20, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 25, 25, 25, 25, 25, 25, 25, 30, 35, 40, 45, 50, 55, 60,  65],
 [ 30, 30, 30, 30, 30, 30, 30, 30, 35, 40, 45, 50, 55, 60,  65]]
```

Como puedes observar, el punto de luz está en la posición (4,2), ya que es la posición de la matriz que presenta un menor valor. También puedes ver la organización del resto de la matriz alrededor de dicho punto, con valores crecientes en cuadrados concéntricos.

Todas las posibles matrices tendrán la estructura de la del ejemplo aunque, evidentemente, en otros casos el punto de luz podría estar situado en cualquier posición de la matriz y los valores contenidos en la matriz podrán variar en función de la luminosidad original.

Has de diseñar un algoritmo de *divide y vencerás* que, dada una matriz como la descrita, averigüe de la manera más eficiente posible la posición del punto de luz. Se pìde:

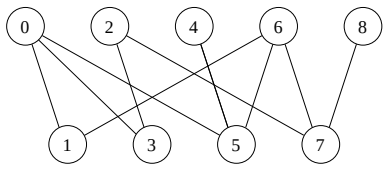
a) Diseña un algoritmo recursivo mediante la técnica de divide y vencerás que permita resolver este problema.

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

Un grafo no dirigido $G = (V, E)$ es bipartito si su conjunto de vértices V puede partirse en dos subconjuntos disjuntos V_1 y V_2 de modo que sólo haya aristas de la forma (u, v) con $u \in V_1$ y $v \in V_2$ (es decir, no puede haber ni una sóla arista entre vértices de un mismo subconjunto).

Por ejemplo, el siguiente grafo es bipartito:



ya que si dividimos V en los dos subconjuntos $V_1 = \{0, 2, 4, 6, 8\}$ y $V_2 = \{1, 3, 5, 7\}$, todas las aristas de E unen vértices de V_1 con vértices de V_2.

Debes desarrollar un algoritmo de *búsqueda con retroceso* que, a partir de $G = (V, E)$, devuelva, si existe, la bipartición de V en dos conjuntos. 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.

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

En un concurso de programación hay n participantes que debemos organizar en equipos de 3, 2 o 1 persona. De cada persona conocemos su grado de afinidad con los demás participantes, recogida en una matriz A, de forma que $A[i, j]$ nos indica la afinidad entre la persona i y la j (no necesariamente simétrica y que puede ser negativa si se llevan mal).

Queremos desarrollar un *algoritmo voraz* que organice los equipos de manera que se máximice la suma de afinidades entre los miembros de los equipos formados.

Un ejemplo para $n = 10$ participantes podría venirnos dado a partir de la siguiente matriz de afinidades:

A	1	2	3	4	5	6	7	8	9	10
1	-	-3	4	11	6	3	5	7	-8	-5
2	4	-	0	3	2	4	12	0	-1	5
3	-1	0	-	6	9	8	7	-10	-9	3
4	9	3	1	-	-2	-5	6	8	3	0
5	0	-2	15	-1	-	3	-7	-5	0	1
6	-2	3	4	1	6	-	-5	-3	0	-7
7	0	11	3	-1	2	-3	-	7	6	9
8	5	3	4	9	-1	-3	-8	-	4	3
9	3	5	0	-2	1	7	0	2	-	-3
10	-3	8	5	-4	-7	5	7	5	-9	-

Una posible solución para esta matriz sería hacer los grupos ((1,3,7), (9,6), (2,4), (8,5), (10)), cuya suma de afinidades sería: $(4 + 5 - 1 + 7 + 0 + 3) + (7 + 0) + (3 + 3) + (-1 - 5) + (0) = 18 + 7 + 6 - 6 + 0 = 25$. Contesta a las siguientes cuestiones:

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 propiedad debe cumplir una solución factible para ser considerada óptima.

b) Para resolver este problema, se plantea una estrategia voraz basada en ir construyendo equipos de manera incremental (distinguiendo en el proceso entre equipos definitivos y equipos candidatos) y la idea es ir modificando los equipos candidatos para, en un momento dado, pasarlos a definitivos. La estrategia es la siguiente: "Inicialmente cada persona es un equipo candidato (de un sólo miembro). Mientras queden al menos 3 participantes pendientes de clasificación definitiva, buscamos en A el valor máximo de $A[i, j]$ entre los participantes i y j que no estén en equipos definitivos y que no pertenezcan al mismo equipo candidato, y fusionamos los equipos candidatos a los que pertenecen i y j en un nuevo equipo candidato, teniendo en cuenta que si formamos un equipo de 3 personas, el equipo pasa de candidato a definitivo. En el momento en que queden menos de 3 participantes por considerar, se fusionan como un equipo definitivo y acabamos."

   Con respecto a esta estrategia:

   1. ¿Garantiza encontrar una solución factible? Justifica tu respuesta (si la respuesta es no, debes incluir un contraejemplo).
   2. ¿Encuentra siempre una solución óptima? Justifica la respuesta (si la respuesta es no, debes incluir un contraejemplo).
   3. ¿Cuál sería la complejidad temporal de la estrategia para implementarla de la manera más eficiente posible (indica para ello las estructuras de datos concretas que utilizarías, describiendo los detalles que justifiquen el coste señalado)?

c) Planteamos ahora esta otra estrategia voraz basada en construir los equipos directamente a partir de los participantes más afines aún no asignados a ningún equipo, de manera que en el momento en que se construye un equipo este es definitivo. La estrategia es la siguiente: "Inicialmente ningún participante forma parte de ningún equipo. Mientras queden al menos 3 participantes pendientes de asignar, buscamos el participante i cuyas suma de valores $A[i, j] + A[i, k]$ sea máxima, donde i, j y k deben ser participantes aún no asignados, y creamos un equipo con esos tres participantes. En el momento en que queden menos de 3 participantes por considerar, se crea un equipo con ellos y acabamos." Responde a las mismas preguntas (1, 2 y 3) del anterior apartado para esta nueva estrategia.

4. Programación dinámica — 3,5 puntos

Se desea hacer una copia de seguridad de N ficheros en un DVD con capacidad (entera) de M megas de manera que podamos llenarlo al máximo, pero siempre y cuando no metamos más de F ficheros. Cada fichero tiene un tamaño (entero) de $t(i)$ megas ($1 \leq i \leq N$).

Deseamos diseñar un algoritmo mediante la técnica de *programación dinámica* que resuelva el problema de obtener el valor de la máxima ocupación en el DVD que podamos conseguir con las condiciones fijadas. Se pide:

a) Formaliza el problema en términos de optimización.

b) Diseña una ecuación recursiva que calcule el valor de la máxima ocupación posible del DVD con un máximo de F ficheros, indicando cuál debe ser la llamada que se haga a dicha ecuación para resolver el problema.

c) Diseña un algoritmo iterativo que recorra el anterior grafo y devuelva el valor solicitado empleando la menor ocupación espacial (en términos asintóticos) posible. Indica y justifica sus costes espacial y temporal.

Examen segunda convocatoria de Esquemas Algorítmicos (IG24)

2 de septiembre de 2008

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 *desordenado* de n enteros, diseña un algoritmo mediante la técnica de *divide y vencerás* que devuelva `true` si la suma de los k menores elementos del vector es mayor que un valor dado V y `false` en caso contrario.

Se pide:

a) Diseña un algoritmo recursivo *mediante la técnica de divide y vencerás* que permita resolver este problema con un coste temporal *esperado* de $O(n)$.

Pistas: para facilitar la elaboración del algoritmo, puedes asumir que dispones de la función `partition` empleada en el algoritmo `quicksort`. La función de *divide y vencerás* que desarrolles no tiene porque calcular directamente el resultado pedido, sino que puede devolver algún resultado intermedio que permita, una vez acabado su cálculo, y mediante otra función, obtener la solución. En cualquier caso, la estrategia principal de resolución deberá ser la de *divide y vencerás* y todas las funciones necesarias para el cálculo solicitado deberás incluirlas (la función `partition` no es necesario que la incluyas).

b) Analiza y justifica la complejidad temporal y espacial del algoritmo desarrollado en el apartado anterior en el mejor y el peor caso. Señala ante qué tipo de entradas puede el algoritmo comportarse en el mejor de los casos y en qué condiciones en el peor.

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

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 sóla vez*. Para ello 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 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

El paintball es un juego de guerra entre dos equipos armados con balas de pintura. Participamos en una competición en la que el equipo enemigo ha tomado n objetivos y queremos reconquistarlos. En cada uno de los i objetivos ($1 \leq i \leq n$) el enemigo ha dejado a x_i personas defendiéndolo. Nuestro equipo está formado por n comandos de intervención, cada uno de los cuales tiene y_i personas ($1 \leq i \leq n$). Nos interesa reconquistar la mayor cantidad posible de objetivos. Para *garantizar* el éxito de la intervención en un objetivo, es necesario que el comando de ataque tenga el menos tantos efectivos como el número de defensores.

Debes diseñar una *estrategia voraz* lo más eficiente posible que nos permita conquistar el mayor número de objetivos, describiéndola con claridad e indicando y justificando su coste temporal.

4. Programación dinámica — 3,5 puntos

Se necesita llenar la bodega de un barco con B millones de litros de agua procedentes de una planta desalinizadora. La capacidad de producción de agua de la planta supera esa cantidad. Para cargar la bodega del barco, la planta dispone de un juego de C cubas (numeradas de 1 a C). La cuba i-ésima tiene una capacidad de l_i millones de litros, siendo dicha cantidad un número entero. Sólo es posible trasvasar agua de la planta a la bodega del barco utilizando las cubas (debe llenarse completamente cada cuba usada y volcarse toda el agua de la cuba en la bodega), pudiendo emplearse más de una vez una misma cuba. Si también conocemos el tiempo t_i que utilizamos en el proceso de llenado, traslado y vaciado de cada cuba i ($1 \leq i \leq C$), ¿cuál es el menor tiempo que podemos emplear para llenar completamente la bodega del barco utilizando las cubas disponibles?

Se pide:

a) Formaliza el problema en términos de optimización,

b) plantea la ecuación recursiva de *programación dinámica* que calcula el menor tiempo con el que llenar la bodega, 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 $B = 13$, $C = 5$, $l = [3, 2, 4, 6, 1]$ y $t = [5, 6, 7, 10, 1]$, y

d) diseña un algoritmo iterativo de *programación dinámica* lo más eficiente posible que recorra el anterior grafo y devuelva el tiempo mínimo con que puede llenarse la bodega y las cubas empleadas para realizar dicho trasvase, indicando y justificando sus costes espacial y temporal.