Análisis de algoritmos y Estructuras de datos
27 de febrero de 2006 (Mañana)

Prueba individual

Nombre: **Grupo:**

Análisis de algoritmos

Los dos siguientes algoritmos calculan, a partir de una matriz M de valores booleanos y de talla $n \times n$, el cierre transitivo de dicha matriz:

```
def cierre_transitivo_1(M):
    n = len(M)
    R = [ [ M[i][j] for j in xrange(n)] for i in xrange(n)]
    for k in xrange(n):
        for i in xrange(n):
            for j in xrange(n):
                R[i][j] = R[i][j] or (R[i][k] and R[k][j])
    return R
```

```
def cierre_transitivo_2(M):
    n = len(M)
    R = [ [ M[i][j] for j in xrange(n)] for i in xrange(n)]
    for k in xrange(n):
        for i in xrange(n):
            if R[i][k]:
                for j in xrange(n):
                    if R[k][j]: R[i][j] = True
    return R
```

Indica y justifica cuál es el coste temporal 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.

Prueba colectiva

Grupo:
Asistentes:

1. MFset modificado

Deseamos implementar un MFset que en todo instante nos permita conocer, dado un elemento cualquiera, el número total de elementos que pertenecen al subconjunto en el que está incluido. ¿Cómo modificarías un MFset para poder ofrecer esta nueva operación con el menor coste temporal posible? Detalla como afectaría a las tres operaciones básicas sobre MFsets (creación de un nuevo subconjunto a partir de un elemento, obtención del representante de un subconjunto dado un elemento del mismo y fusión de dos subconjuntos dados un elemento de cada uno de ellos) y cual es el coste temporal y espacial de la operación suma.

2. Análisis de algoritmos

Una empresa de seguridad ha instalado un sistema de reconocimiento automático de placas de matrícula en la puerta de una fábrica. El primer día registró la entrada de n vehículos y almacenó en un vector a sus matrículas. El segundo día volvió a registrar la entrada de n vehículos y almacenó las matrículas en un segundo vector b. El jefe de seguridad quiso saber si los dos días entraron exactamente los mismos vehículos. Pensó que hacer la comprobación manualmente supondría una inversión excesiva de tiempo, así que encargó al machaca-programador del departamento de informática que diseñara una función en Python para resolver el problema. Esta es la solución que ideó:

```
1  def son_las_mismas(a, b):
2      for i in xrange(len(a)):
3          esta_lo_que_busco = False
4          for j in xrange(len(b)):
5              if a[i] == b[j]:
6                  esta_lo_que_busco = True
7                  break
8          if not esta_lo_que_busco:
9              return False
10     return True
```

Se pide:

a) Un análisis del coste temporal asintótico para el mejor y el peor de los casos (indicando expresamente cuando se produce cada uno de ellos). Expresad el coste en función de n, el tamaño de los vectores a y b.

b) El método *sort* ordena de menor a mayor el contenido de un vector (por ejemplo, si a es un vector, $a.sort()$ lo ordena) en tiempo $O(n \lg n)$ y $\Omega(n)$. Sabiendo esto, ¿podéis diseñar un algoritmo más eficiente? Si es así, describidlo (usando Python o como prefiráis, pero con absoluta claridad) y analizad su coste temporal asintótico en el mejor y en el peor de los casos.

Análisis de algoritmos y Estructuras de datos
27 de febrero de 2006 (Tarde)

Prueba individual

Nombre: **Grupo:**

Análisis de algoritmos

Los dos siguientes algoritmos calculan, a partir de un vector de enteros positivos a, un vector de igual talla cuyo elemento de índice i contiene $\sum_{0 \leq i} a[i]$:

```
1 def vector_sumas_1(a):
2     b = [0] * len(a)
3     for i in xrange(len(a)):
4         s = 0
5         for k in xrange(i+1):
6             s += a[k]
7         b[i] = s
8     return b
```

```
1 def vector_sumas_2(a):
2     b = [0] * len(a)
3     b[0] = a[0]
4     for i in xrange(1,len(a)):
5         b[i] = b[i-1] + a[i]
6     return b
```

Indica y justifica cuál es el coste temporal 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.

Prueba colectiva

Grupo:
Asistentes:

1. Selección/diseño de una estructura de datos

Un científico necesita disponer de una estructura de datos T en la que almacenar un máximo de m valores enteros para realizar una serie de simulaciones. Un mismo número entero puede aparecer más de una vez, habiendo como máximo n números enteros distintos (n será normalmente mucho menor que m). La estructura debe ser lo más eficiente posible en las siguientes operaciones:

1. añadir un número a T,
2. consultar el valor mínimo de T,
3. eliminar el valor mínimo de T (si hubiera más de una aparición de dicho número, eliminar una cualquiera de ellas),

Debes señalar brevemente como implementarías esta estructura de datos y cómo y con que coste temporal se podría ejecutar cada una de las tres operaciones anteriormente citadas si nos imponen una restricción: *el espacio que podemos emplear está acotado por* $O(n)$ (recuerda que m podría ser mucho mayor que n).

2. Análisis de algoritmos

Una empresa de transporte tiene que enviar n contenedores de distintos pesos a un mismo destino utilizando su flota de camiones. Los camiones son idénticos y soportan la misma carga máxima, C. Cada contenedor tiene un peso $p_0, p_1, \ldots, p_{n-1}$ (suponemos que $p_i \leq C$ para todo i). A la empresa le gustaría organizar los contenedores de forma que se utilice el menor número posible de camiones en el transporte.

El siguiente algoritmo proporciona una buena aproximación a cuál es dicho número de camiones. Inicialmente se ordenan los contenedores de mayor a menor peso y se considera que como mínimo será necesario disponer de tantos camiones como contenedores hay, con una capacidad de carga máxima (C) (líneas 2 a 4). La estrategia seguida a continuación considera cada contenedor e intenta introducirlo en el camión en el que queda menos capacidad de carga disponible. Tras encontrar un camión en el que cabe el contenedor e introducirlo, se reordenan los camiones de forma que siempre estén organizados de menor a mayor capacidad de carga (líneas 10 a 12). Finalmente, y tras procesar todos los contenedores, se cuenta y devuelve el número mínimo de camiones necesarios (líneas 14 a 20):

```
1  def num_camiones_minimo(p, C):
2      P = sorted(p)
3      P.reverse()
4      capac_disp_camion = [C]*len(p)
5      for i in xrange(len(p)):
6          for j in xrange(len(p)):
7              if capac_disp_camion[j] >= P[i]:
8                  capac_disp_camion[j] -= P[i]
9                  k=j
10                 while k>1 and capac_disp_camion[k]<capac_disp_camion[k-1]:
11                     capac_disp_camion[k],capac_disp_camion[k-1]=capac_disp_camion[k-1],capac_disp_camion[k]
12                     k-=1
13                 break
14     num_cam=0
15     for i in xrange(len(p)):
16         if capac_disp_camion[i] < C:
17             num_cam +=1
18         else:
19             break
20     return num_cam
```

Se pide un análisis del coste temporal asintótico para el mejor y el peor de los casos (indicando expresamente cuando se produce cada uno de ellos). Expresad el coste en función de n, el número de contenedores.

Algoritmos voraces
15 de marzo de 2006 (Mañana)

Prueba individual

Nombre: **Grupo:**

Código de Huffman — Modelo 1

En un partido de voleibol se pueden producir 6 distintos resultados: $(3,0), (3,1), (3,2), (2,3), (1,3), (0,3)$. La siguiente tabla recoge, para cada posible resultado, el número total de partidos finalizados en la actual liga masculina de voleibol:

Resultado	$(3,0)$	$(3,1)$	$(3,2)$	$(2,3)$	$(1,3)$	$(0,3)$
Número de partidos	155	90	105	80	65	100

Deseamos encontrar la codificación binaria óptima de los resultados mediante el algoritmo de Huffman tomando la anterior tabla como medida de las frecuencias de cada resultado. Se pide:

a) Dibuja el árbol binario de decodificación que construiría el algoritmo de Huffman basándose en los anteriores datos.

b) Según dicho árbol, indica qué código binario le correspondería a cada resultado posible.

c) El anterior árbol es devuelto por el algoritmo mediante una representación de listas de listas: escribe la lista correspondiente a dicho árbol.

Código de Huffman — Modelo 2

En un partido de voleibol se pueden producir 6 distintos resultados: $(3,0), (3,1), (3,2), (2,3), (1,3), (0,3)$. La siguiente tabla recoge, para cada posible resultado, el número total de partidos finalizados en la actual liga femenina de voleibol:

Resultado	$(3,0)$	$(3,1)$	$(3,2)$	$(2,3)$	$(1,3)$	$(0,3)$
Número de partidos	220	50	85	30	90	120

Deseamos encontrar la codificación binaria óptima de los resultados mediante el algoritmo de Huffman tomando la anterior tabla como medida de las frecuencias de cada resultado. Se pide:

a) Dibuja el árbol binario de decodificación que construiría el algoritmo de Huffman basándose en los anteriores datos.

b) Según dicho árbol, indica qué código binario le correspondería a cada resultado posible.

c) El anterior árbol es devuelto por el algoritmo mediante una representación de listas de listas: escribe la lista correspondiente a dicho árbol.

Algoritmos voraces
15 de marzo de 2006 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Problema

Un mensajero que trabaja para una compañía de reparto de paquetes postales cobra una cantidad fija por cada entrega que realiza a lo largo de un día. Su trabajo siempre es el mismo: sale del almacén con el paquete correspondiente, llega al destino a la hora fijada para la entrega y vuelve al almacén, donde esperará hasta que le toque realizar la siguiente entrega por el mismo procedimiento.

El mensajero puede elegir los paquetes que entregará durante el día. Para ello, quiere hacer uso de la información que la compañia postal facilita al comienzo de cada jornada: de los n envíos que hay que realizar ese día (y que deben efectuar entre todos los mensajeros de la empresa) conoce, para cada envío i (están numerados entre 1 y n) la hora de la entrega del paquete (h_i) y los minutos que cuesta desplazarse hasta el lugar de destino del paquete (t_i) y regresar al almacén (t'_i). Por tanto, la información disponible para los n paquetes será $p = \{(h_1, t_1, t'_1), (h_2, t_2, t'_2), \ldots, (h_n, t_n, t'_n)\}$.

Diseña una estrategia voraz que permita al mensajero elegir aquella combinación de repartos que contenga la mayor cantidad posible de entregas durante el día para así cobrar más.

Ejemplo: Dado un día con 8 repartos con la siguiente información: $p = \{$(9:00,17,20), (12:15,13,13), (10:45,28,30), (14:25,30,25), (11:30,8,8), (16:30,20,22), (8:30,20,20), (12:00,15,10)$\}$, una solución factible (aunque no la óptima) sería la formada por las entregas ((8:30,20,20), (10:45,28,30), (12:15,13,13), (16:30,20,22)), lo que le permitiría realizar 4 entregas ese día. Tened cuidado con la interpretación correcta del triplete de valores correspondiente a un paquete: en la entrega (9:00,17,20), las 9:00 es la hora de entrega, el mensajero invierte 17 minutos en desplazarse para entregar el paquete y otros 20 para volver al almacén, por lo que tendría que salir del almacén a las 8:43 y estaría de vuelta a las 9:20. Con esta restricción no es posible, por ejemplo, que si el mensajero realiza la entrega (12:00,15,10) pueda realizar la entrega (12:15,13,13).

Se pide:

a) Formalizad el conjunto de soluciones factibles y la función objetivo para el planteamiento del problema.

b) Proponed una estrategia voraz que devuelva la solución óptima para el problema ante cualquier entrada válida.

c) Diseñad un algoritmo que siga dicha estrategia.

d) Indicad (y justificad) cuál es el coste temporal de la estrategia (y del algoritmo) propuestos.

e) Suponiendo que al mensajero le interesa, en el caso de haber más de una combinación de paquetes óptima, quedarse con aquella que le permita acabar lo antes posible la jornada laboral, ¿funcionaría correctamente la estrategía que habéis diseñado? Justificad la respuesta y, si es negativa, indicad (comentándola) si es posible emplear alguna otra estrategia voraz para resolver la variante planteada. Responded a las mismas cuestiones para el caso en el que lo que quiere el mensajero es, ante varías combinaciones óptimas, seleccionar la que le permita comenzar lo más tarde posible.

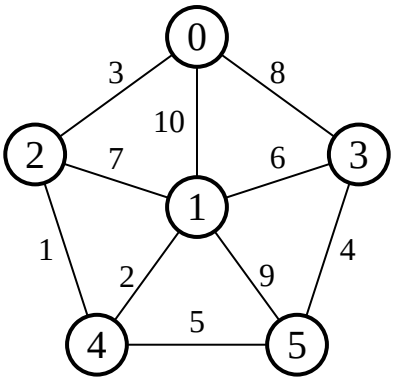
Algoritmos voraces
15 de marzo de 2006 (Tarde)

Prueba individual

Nombre: **Grupo:**

Algoritmo de Prim **Modelo 1**

Dado el siguiente grafo no dirigido y ponderado,

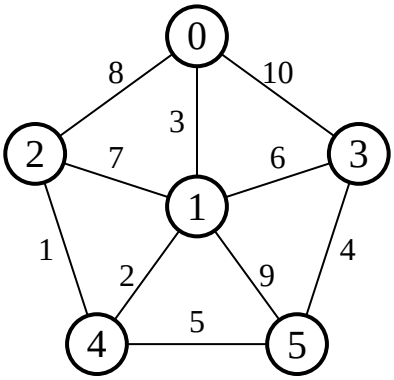


a) Indica cuál sería el árbol de recubrimiento mínimo que devolvería el algoritmo de Prim si el vértice elegido en primer lugar es el 0 (escribe las aristas en el orden en que las devolvería el algoritmo).

b) Haz una traza, iteración a iteración, del algoritmo `prim2.py` (pág. 5-25 y 5-26). Para ello, rellena la siguiente tabla:

	touched	*MST*	*frontier*
0			
1			
2			
3			
4			
5			
6			
7			
...			

Dado el siguiente grafo no dirigido y ponderado,



a) Indica cuál sería el árbol de recubrimiento mínimo que devolvería el algoritmo de Prim si el vértice elegido en primer lugar es el 5 (escribe las aristas en el orden en que las devolvería el algoritmo).

b) Haz una traza, iteración a iteración, del algoritmo `prim2.py` (pág. 5-25 y 5-26). Para ello, rellena la siguiente tabla:

	touched	*MST*	*frontier*
0			
1			
2			
3			
4			
5			
6			
7			
...			

Algoritmos voraces
15 de marzo de 2006 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Problema

Un granjero virtual tiene que mantener una granja formada por n tamagotchis de distintas formas y tamaños. El granjero dispone de una tabla $t = (t_1, t_2, \ldots, t_n) \in \mathbb{R}^n$ en la que, para cada tamagotchi i ($1 \leq i \leq n$), el valor t_i indica la ración de alimentos y de mimos (que son coincidentes) que da cada día al tamagotchi i para mantenerle muy contento.

Por un error informático, que el granjero no podrá reparar hasta el día siguiente, se ha producido una reducción de la cantidad total de alimentos y de mimos de que dispone para repartir entre sus tamagotchis (la reducción ha sido muy importante en los alimentos y algo menos en los mimos). El granjero está desesperado, ya que los tamagotchis son muy sensibles y si cada día no reciben una cantidad mínima de alimentos y/o mimos mueren.

La cantidad total de alimentos de la que dispone el granjero para pasar el día es $A \in \mathbb{R}$ y la de mimos es $M \in \mathbb{R}$ (donde $M \geq 2 * A$). Para sobrevivir durante un día un tamagotchi necesita recibir al menos una de dos opciones: o bien la mitad de la ración de alimentos que le corresponde habitualmente ($1/2 * t_i$) o bien una ración completa de los mimos que recibe diariamente (t_i). Sabiendo esto, diseña una estrategia voraz que permita al granjero repartir la comida y los mimos de que dispone entre los tamagotchis de forma que consiga salvar al mayor número posible de ellos.

Ejemplo: Raciones de comida y mimos que reciben habitualmente 5 tamagotchis: $t = [10, 19, 15, 12, 6]$. Alimentos y mimos disponibles para pasar el día de la avería: $A = 13$ y $M = 30$. Una solución factible sería ((0,1), (0.5,0), (0,0), (0,1), (0.5,0)), donde ha gastado $0.5 * 19 + 0.5 * 6 = 12.5$ alimentos y $1 * 10 + 1 * 12 = 22$ mimos. El número de tamagotchis que sobrevivirían con esta solución es 4.

Se pide:

a) Formalizad el conjunto de soluciones factibles e indicad, a partir de él, cómo se calcularía el valor de la función objetivo sobre una solución factible (esto no es preciso formalizarlo).

b) Proponed una estrategia voraz que devuelva la solución óptima para el problema ante cualquier entrada válida.

c) Diseñad un algoritmo que siga dicha estrategia. El algoritmo deberá devolver, para cada tamagotchi, el porcentaje (sobre la ración de alimentos y de mimos que le corresponden habitualmente) que recibirá para alcanzar la solución óptima.

d) Indicad (y justificad) cuál es el coste temporal de la estrategia (y del algoritmo) propuestos.

e) Suponiendo que la ración habitual de alimentos y mimos de todos los tamagotchi fuera la misma, ¿podríais proponer una estrategia más sencilla (y con menor coste temporal) para resolver el problema? ¿cuál sería su coste? En este supuesto, si sólo estuvierais interesados en saber el número de tamagotchis supervivientes, ¿cómo y con coste temporal podríais averiguarlo?

Programación dinámica
4 de mayo de 2006 (Mañana)

Prueba individual

Nombre: **Grupo:**

Problema de la mochila discreta **Modelo 1**

Dada una mochila con capacidad para cargar $W = 4$ unidades de peso y $N = 6$ objetos que podemos cargar en ella, con pesos $w = [1, 3, 1, 2, 4, 3]$ y valores $v = [8, 25, 12, 19, 32, 23]$, se pide:

a) Dibuja, aprovechando la siguiente cuadrícula, el *grafo de dependencias extendido* 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 `iterative_knapsack_profit` (pág 7-20). 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`.

c) ¿Cuál es el valor que devuelve el algoritmo indicado? ¿Qué resultado devolvería el algoritmo que calcula la solución óptima del problema `knapsap` (pág 7-22) (para este no es preciso que realices la traza)? ¿Qué significa este último resultado?

Problema de la mochila discreta **Modelo 2**

Dada una mochila con capacidad para cargar $W = 4$ unidades de peso y $N = 6$ objetos que podemos cargar en ella, con pesos $w = [1, 3, 2, 1, 4, 3]$ y valores $v = [8, 25, 16, 12, 32, 23]$, se pide:

a) Dibuja, aprovechando la siguiente cuadrícula, el *grafo de dependencias extendido* 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 `iterative_knapsack_profit` (pág 7-20). 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`.

c) ¿Cuál es el valor que devuelve el algoritmo indicado? ¿Qué resultado devolvería el algoritmo que calcula la solución óptima del problema `knapsap` (pág 7-22) (para este no es preciso que realices la traza)? ¿Qué significa este último resultado?

Programación dinámica
4 de mayo de 2006 (Mañana)

Prueba colectiva

Grupo:
Asistentes:

Camino más corto formado por k aristas en un grafo

La siguiente ecuación recursiva (pág 7-37) permite plantear la resolución del problema del camino más corto formado por k aristas en un grafo que empieza en un vértice s y acaba en un vértice t:

$$D(v,k) = \begin{cases} 0, & \text{si } v = s \text{ y } k = 0; \\ +\infty, & \text{si } v \neq s \text{ y } k = 0; \\ +\infty, & \text{si } \nexists (u,v) \in E \text{ y } k > 0; \\ \min\limits_{(u,v)\in E} \left(D(u,k-1) + d(u,v)\right), & \text{en otro caso.} \end{cases}$$

Se pide:

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

b) Escribid el algoritmo iterativo *que permite obtener la secuencia de vértices que forman el camino de distancia mínima* con k aristas entre s y t, indicando su coste temporal y espacial.

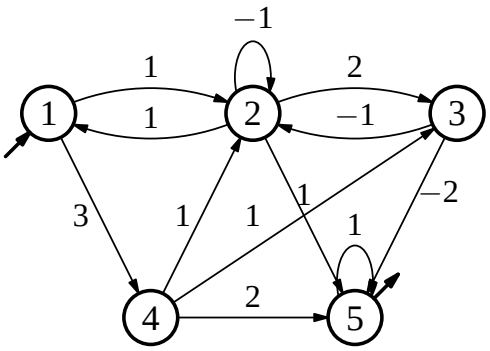
Programación dinámica
4 de mayo de 2006 (Tarde)

Prueba individual

Nombre: **Grupo:**

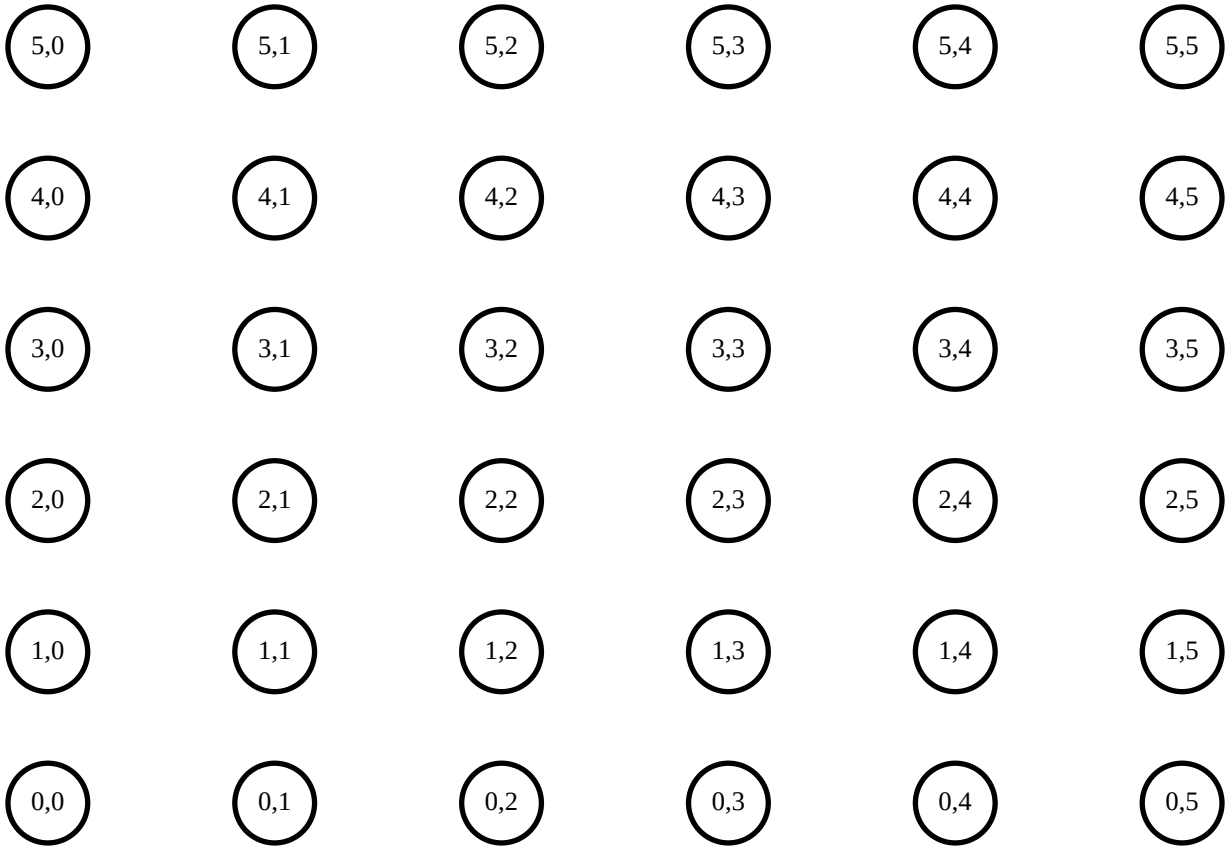
Problema del camino más corto formado por k aristas en un grafo **Modelo 1**

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:



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):

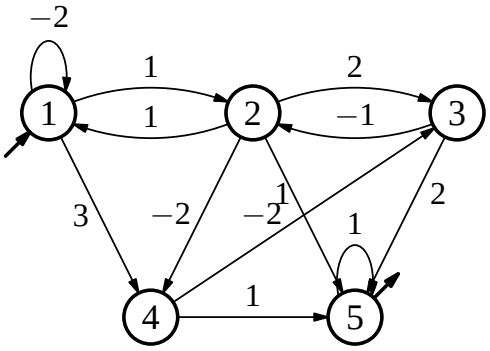


b) Haz una traza del algoritmo `k_edges_shortest_distance` (pág 7-38). Para ello, escribe arriba de cada uno de los estados del grafo del apartado anterior el valor que el algoritmo almacena en el diccionario `D`.

c) ¿Cuál es el valor que devuelve el algoritmo indicado? ¿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 todas las posibles)?

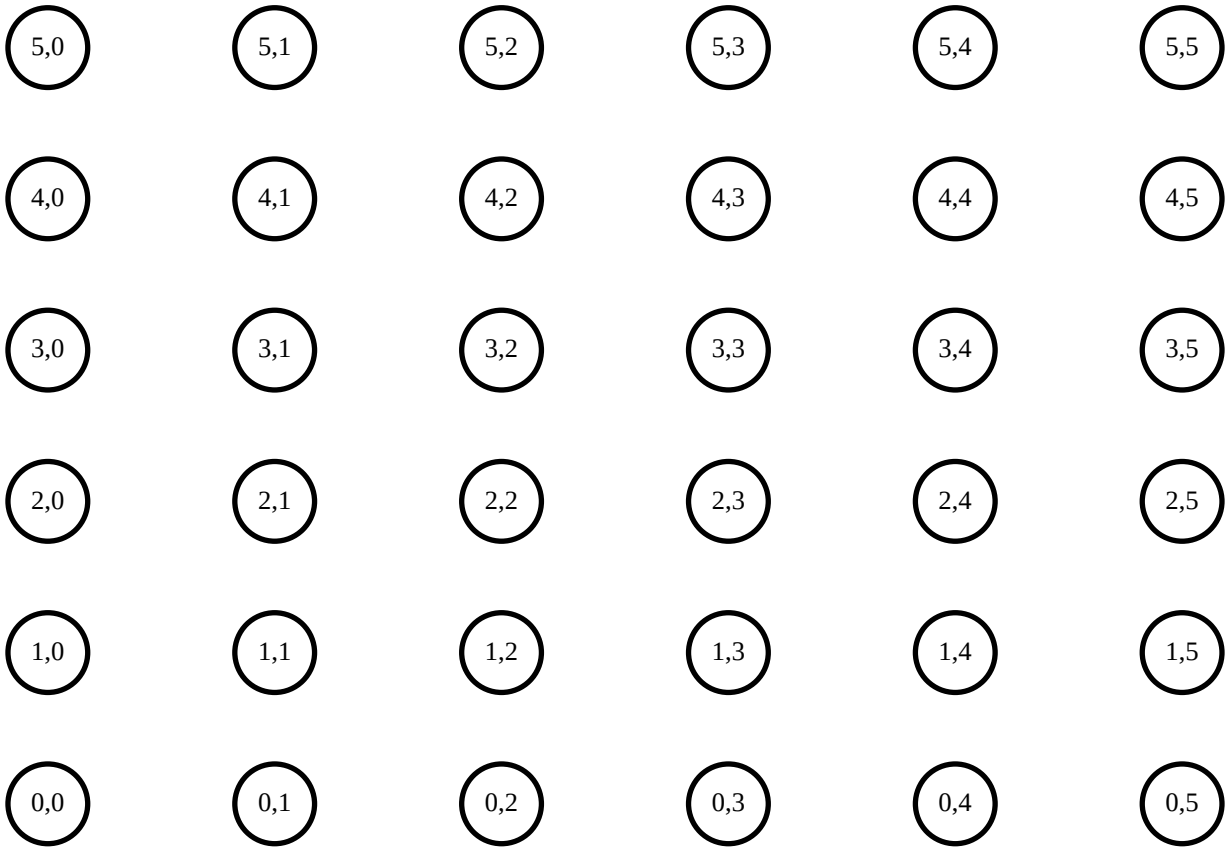
Problema del camino más corto formado por k aristas en un grafo **Modelo 2**

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:



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 que devuelve el algoritmo indicado? ¿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 todas las posibles)?

Programación dinámica
4 de mayo de 2006 (Tarde)

Prueba colectiva

Grupo:
Asistentes:

Trayecto más barato en el río Congo formado por N embarcaderos

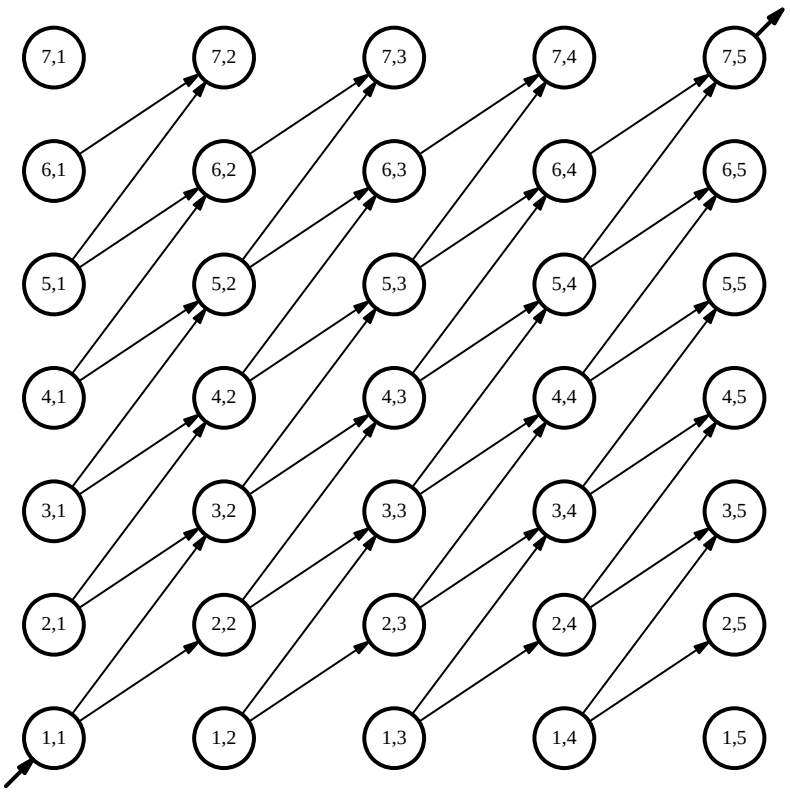
La siguiente ecuación recursiva permite plantear la resolución del problema del trayecto más barato en el río Congo que empieza en el embarcadero 1 y acaba en el embarcadero E y que está formado por *exactamente* N *embarcaderos*:

$$C(i,l) = \begin{cases} 0, & \text{si } i = 1 \text{ y } l = 1; \\ +\infty, & \text{si } i = 1 \text{ y } l \neq 1; \\ +\infty, & \text{si } i \neq 1 \text{ y } l = 1; \\ C(1, l-1) + c(1,2), & \text{si } i = 2 \text{ y } l > 1; \\ \min(C(i-2, l-1) + c(i-2, i), C(i-1, l-1) + c(i-1, i)), & \text{si } i > 2 \text{ y } l > 1, \end{cases}$$

donde el coste del trayecto con que un turista viaja del primer al último embarcadero pasando por exactamente N embarcaderos de la forma más económica es $C(E, N)$. Se pide:

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

b) Teniendo en cuenta que el grafo de dependencias entre estados asociado a la anterior ecuación recursiva es el siguiente (para un ejemplo con $E = 7$ embarcaderos y $N = 5$ escalas):



Escribid el algoritmo iterativo *con reducción de la complejidad espacial* que permite calcular el valor del trayecto más barato entre el primer y el último embarcadero que pasa por exactamente N embarcaderos. Indicad cuál es su coste temporal y espacial.

Programación dinámica (II)
15 de mayo de 2006 (Mañana, modelo 1)

Prueba individual

Nombre: **Grupo:**

Análisis de un problema **35 puntos**

Dados los números naturales N y K, una descomposición de N en K sumandos positivos es una secuencia n_1, n_2, ..., n_K, tal que

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

Deseamos obtener la descomposición de N en K sumandos de modo que sea máximo su producto

$$\prod_{1 \leq k \leq K} n_k.$$

Por ejemplo, si $N = 6$ y $K = 3$ tenemos las siguientes descomposiciones posibles: $1 + 1 + 4$, $1 + 2 + 3$ y $2 + 2 + 2$. El producto de los sumandos es, respectivamente, $1 \cdot 1 \cdot 4 = 4$, $1 \cdot 2 \cdot 3 = 6$ y $2 \cdot 2 \cdot 2 = 8$. La descomposición óptima es, pues, $2 + 2 + 2$.

La siguiente ecuación recursiva calcula el producto de los sumandos asociados a la descomposición óptima en el problema anterior:

$$P(n,k) = \begin{cases} n, & \text{si } k = 1; \\ 0, & \text{si } k > 1 \text{ y } n = 1; \\ \max\limits_{1 \leq m \leq n-1} P(n-m, k-1) \cdot m, & \text{si } k > 1 \text{ y } n > 1, \end{cases}$$

donde la llamada $P(N, K)$ proporciona dicho valor. Se pide:

a) Representa el grafo de dependencias para la llamada $P(7, 3)$. Indica qué representa un estado (n, k) y un arco entre dos estados (n', k') y (n, k) dentro del grafo.

b) Indica cuál sería el coste temporal y espacial de un algoritmo iterativo diseñado a partir de la ecuación presentada que devolviera *la descomposición óptima*. Justifica brevemente tu respuesta.

c) ¿Crees que sería posible reducir el coste espacial indicado en el apartado anterior si tuviéramos que diseñar un algoritmo iterativo que sólo encontrara *el valor* del producto de la descomposición óptima? Si es así, indica cuál sería dicho coste y justifícalo.

Programación dinámica (II)
15 de mayo de 2006 (Mañana)

Prueba colectiva

Grupo:
Asistentes:
P_2: Peso de este control en la nota de programación dinámica (mín 50 %, máx 80 %):..... %
P_1: Peso del control previo en la nota de programación dinámica (mín 20 %, máx 50 %):..... %
($P_2 + P_1 = 100\,\%$. Si se deja en blanco o no suma 100 %, el peso será del 50 % en cada control.)

Resolución de problemas **65 puntos**

Sea el anterior problema:

****

Dados los números naturales N y K, una descomposición de N en K sumandos positivos es una secuencia n_1, n_2, ..., n_K, tal que

$$N = \sum_{1 \le k \le K} n_k \,.$$

Deseamos obtener la descomposición de N en K sumandos de modo que sea máximo su producto

$$\prod_{1 \le k \le K} n_k.$$

Por ejemplo, si $N = 6$ y $K = 3$ tenemos las siguientes descomposiciones posibles: $1 + 1 + 4$, $1 + 2 + 3$ y $2 + 2 + 2$. El producto de los sumandos es, respectivamente, $1 \cdot 1 \cdot 4 = 4$, $1 \cdot 2 \cdot 3 = 6$ y $2 \cdot 2 \cdot 2 = 8$. La descomposición óptima es, pues, $2 + 2 + 2$.

La siguiente ecuación recursiva calcula el producto de los sumandos asociados a la descomposición óptima en el problema anterior:

$$P(n,k) = \begin{cases} n, & \text{si } k = 1; \\ 0, & \text{si } k > 1 \text{ y } n = 1; \\ \max\limits_{1 \le m \le n-1} P(n-m, k-1) \cdot m, & \text{si } k > 1 \text{ y } n > 1, \end{cases}$$

donde la llamada $P(N, K)$ proporciona dicho valor.

****

Se proponen las dos siguientes modificaciones en el enunciado del problema de la maximización del producto de los sumandos resultantes de la descomposición de un número N:

1. La descomposición no tiene por qué tener un número fijo K de componentes.
2. Ningún sumando puede ser mayor que un valor dado $M < N$.

El resto de condiciones se mantienen.
Con las nuevas condiciones y el ejemplo $N = 10$ y $M = 4$, las siguientes descomposiciones serían válidas: 4+2+1+3, $3 + 3 + 4$ y $1 + 2 + 1 + 1 + 2 + 2 + 1$, mientras que la descomposición $2 + 3 + 5$ no lo sería.

Se pide, *para este nuevo problema*:

a) Formalizad el problema en términos de optimización.

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

c) Representad el grafo de dependencias entre las llamadas de la ecuación recursiva para $N = 6$ y $M = 3$.

d) 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.

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

Programación dinámica (II)

15 de mayo de 2006 (Mañana)

Prueba colectiva opcional

Grupo:
Asistentes:

Recuperación del primer control de programación dinámica — +20 puntos (a añadir a P_1)

Diseñad un algoritmo iterativo que devuelva la descomposición óptima en K sumandos de un número N *a partir del enunciado original del problema*:

****

Dados los números naturales N y K, una descomposición de N en K sumandos positivos es una secuencia n_1, n_2, ..., n_K, tal que

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

Deseamos obtener la descomposición de N en K sumandos de modo que sea máximo su producto

$$\prod_{1 \leq k \leq K} n_k .$$

Por ejemplo, si $N = 6$ y $K = 3$ tenemos las siguientes descomposiciones posibles: $1 + 1 + 4$, $1 + 2 + 3$ y $2 + 2 + 2$. El producto de los sumandos es, respectivamente, $1 \cdot 1 \cdot 4 = 4$, $1 \cdot 2 \cdot 3 = 6$ y $2 \cdot 2 \cdot 2 = 8$. La descomposición óptima es, pues, $2 + 2 + 2$.

La siguiente ecuación recursiva calcula el producto de los sumandos asociados a la descomposición óptima en el problema anterior:

$$P(n,k) = \begin{cases} n, & \text{si } k = 1; \\ 0, & \text{si } k > 1 \text{ y } n = 1; \\ \max\limits_{1 \leq m \leq n-1} P(n-m, k-1) \cdot m, & \text{si } k > 1 \text{ y } n > 1, \end{cases}$$

donde la llamada $P(N, K)$ proporciona dicho valor.

****

Programación dinámica (II)
15 de mayo de 2006 (Tarde, modelo 1)

Prueba individual

Nombre: **Grupo:**

Análisis de un problema **35 puntos**

A lo largo del río Amazonas hay N poblados en los que podemos hacer escala. El poblado i está separado del poblado j por una distancia en kilómetros que podemos conocer consultando el valor $d(i, j)$. Sabemos además que es imposible nadar más de K kilómetros sin parar a descansar en un poblado (suponemos $d(i-1, i) \leq K$, para $1 < i \leq N$). Por cortesía, cada vez que llegamos a un poblado i debemos descansar en él tantos días como nos indica $c(i)$, la costumbre local (donde $2 \leq i \leq N-1$, es decir, no se cuentan los días de estancia en el primer ni en el último poblado).

Se pretende averiguar cual es el mínimo número de días necesario para poder realizar la travesía a nado desde el poblado 1 hasta el N.

Una ecuación recursiva que permite resolver el problema es la siguiente:

$$V(j) = \begin{cases} 0, & \text{si } j = 1; \\ \min\limits_{\substack{1 \leq i < j: \\ d(i,j) \leq K}} V(i) + c(i), & \text{si } j > 1, \end{cases}$$

donde la llamada $V(N)$ proporciona el número mínimo de días. Se pide:

a) Representa el grafo de dependencias para la llamada $V(7)$ con $K = 10$ y las distancias:

$$d = \{(1,2):5, (1,3):8, (1,4):12, (1,5):15, (1,6):17, (1,7):23, (2,3):3, (2,4):7, (2,5):10, (2,6):12,$$
$$(2,7):18, (3,4):4, (3,5):7, (3,6):9; (3,7):16, (4,5):3, (4,6):5, (4,7):11, (5,6):2, (5,7):8, (6,7):6\}$$

Indica qué representa un estado j y un arco entre dos estados i y j dentro del grafo.

b) Indica cuál sería el coste temporal y espacial de un algoritmo iterativo diseñado a partir de la ecuación presentada que devolviera *la secuencia de embarcaderos óptima*. Justifica brevemente tu respuesta.

c) ¿Crees que sería posible reducir el coste espacial indicado en el apartado anterior si tuviéramos que diseñar un algoritmo iterativo que sólo encontrara *el valor* del mínimo número de días necesarios para recorrer el río? Si es así, indica cuál sería dicho coste y justifícalo.

Programación dinámica (II)
15 de mayo de 2006 (Tarde)

Prueba colectiva

Grupo:
Asistentes:
P_2: Peso de este control en la nota de programación dinámica (mín 50 %, máx 80 %):..... %
P_1: Peso del control previo en la nota de programación dinámica (mín 20 %, máx 50 %):..... %
($P_2 + P_1 = 100\,\%$. Si se deja en blanco o no suma 100 %, el peso será del 50 % en cada control.)

Resolución de problemas **65 puntos**

Sea el anterior problema:

****

A lo largo del río Amazonas hay N poblados en los que podemos hacer escala. El poblado i está separado del poblado j por una distancia en kilómetros que podemos conocer consultando el valor $d(i, j)$. Sabemos además que es imposible nadar más de K kilómetros sin parar a descansar en un poblado (suponemos $d(i-1, i) \leq K$, para $1 < i \leq N$). Por cortesía, cada vez que llegamos a un poblado i debemos descansar en él tantos días como nos indica $c(i)$, la costumbre local (donde $2 \leq i \leq N-1$, es decir, no se cuentan los días de estancia en el primer ni en el último poblado).

Se pretende averiguar cual es el mínimo número de días necesario para poder realizar la travesía a nado desde el poblado 1 hasta el N.

Una ecuación recursiva que permite resolver el problema es la siguiente:

$$V(j) = \begin{cases} 0, & \text{si } j = 1; \\ \min\limits_{\substack{1 \leq i < j: \\ d(i,j) \leq K}} V(i) + c(i), & \text{si } j > 1, \end{cases}$$

donde la llamada $V(N)$ proporciona el número mínimo de días.

****

Se proponen las tres siguientes modificaciones en el enunciado del problema del cálculo de la travesía a nado en menos días a través del río Amazonas:

1. Hay que realizar exactamente M escalas (incluyendo el poblado de salida y el de llegada).
2. Como reto personal, el nadador se ha propuesto que, una vez sale de un poblado, no puede parar en el inmediatamente posterior.
3. No conocemos las distancias $d(i, j)$ entre poblados. Sin embargo, para cada poblado $1 < j \leq N$, sí que nos indican $p(j)$, que es el poblado más lejano desde el que podemos llegar nadando sin parar hasta j. Por ejemplo, $p(7) = 3$ significa que al poblado 7 podemos llegar directamente como muy lejos desde el poblado 3, (también, por tanto, desde todos los que hay entre el 3 y el 7, como el 4, el 5 o el 6) pero no es posible llegar al 7 desde el 1 o el 2 sin hacer ninguna escala.

Con las nuevas condiciones y el ejemplo $N = 8$, $M = 4$ y $p = (-, 1, 1, 1, 2, 4, 4, 4)$, las siguientes travesías serían válidas: $(1, 3, 5, 8)$ y $(1, 4, 6, 8)$. Sin embargo, no serían válidas: $(1, 4, 8)$, por la restricción 1 (hay 3 escalas, no 4); $(1, 2, 5, 8)$, por la restricción 2 (no se puede parar en el 2 si el anterior es el 1); ni $(1, 3, 6, 8)$, por la restricción 3 ($p(6) = 4$, por lo que no se puede ir del 3 al 6).

Se pide, *para este nuevo problema*:

a) Formalizad el problema en términos de optimización.

b) Plantead la ecuación recursiva de programación dinámica que calcula el número de días mínimo para realizar la travesía óptima, indicando cuál debe ser la llamada que se efectue a dicha función para obtener el valor óptimo.

c) Representad el grafo de dependencias entre las llamadas de la ecuación recursiva para $N = 6$, $M = 3$ y $p = (-, 1, 1, 1, 2, 4)$.

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

e) ¿Sería posible reducir el coste espacial indicado en el apartado anterior si hubiera que diseñar un algoritmo iterativo que sólo encontrara *el valor* del mínimo número de días necesarios para recorrer el río? Si es así, indicad cuál sería dicho coste y justificadlo.

Programación dinámica (II)

15 de mayo de 2006 (Tarde)

Prueba colectiva opcional

Grupo:
Asistentes:

Recuperación del primer control de programación dinámica — +20 puntos (a añadir a P_1)

Diseñad un algoritmo iterativo que devuelva la travesía óptima en el río Amazonas *según la definición del enunciado original del problema*:

****

A lo largo del río Amazonas hay N poblados en los que podemos hacer escala. El poblado i está separado del poblado j por una distancia en kilómetros que podemos conocer consultando el valor $d(i, j)$. Sabemos además que es imposible nadar más de K kilómetros sin parar a descansar en un poblado (suponemos $d(i-1, i) \leq K$, para $1 < i \leq N$). Por cortesía, cada vez que llegamos a un poblado i debemos descansar en él tantos días como nos indica $c(i)$, la costumbre local (donde $2 \leq i \leq N-1$, es decir, no se cuentan los días de estancia en el primer ni en el último poblado).

Se pretende averiguar cual es el mínimo número de días necesario para poder realizar la travesía a nado desde el poblado 1 hasta el N.

Una ecuación recursiva que permite resolver el problema es la siguiente:

$$V(j) = \begin{cases} 0, & \text{si } j = 1; \\ \min\limits_{\substack{1 \leq i < j: \\ d(i,j) \leq K}} V(i) + c(i), & \text{si } j > 1, \end{cases}$$

donde la llamada $V(N)$ proporciona el número mínimo de días.

****

Primer control parcial de Esquemas Algorítmicos (IG24)

8 de abril de 2006

1. Algoritmos voraces — 5 puntos

Un nadador profesional se prepara para acudir a las olimpiadas. El nadador es capaz de nadar con solvencia en n pruebas. Para cada prueba tiene la estimación del esfuerzo que necesita realizar para quedar en tercer lugar: $e = (e_1, e_2, \ldots, e_n) \in \mathbb{N}^n$. Quedar en segundo lugar en cada prueba le costará el doble del esfuerzo que hacerlo en tercer lugar y para ganar la prueba deberá realizar el triple de esfuerzo que para quedar tercero. El premio que recibirá por cada prueba ganada es de 100.000 euros, 40.000 por cada segundo puesto y 10.000 por cada tercera posición.

El nadador sabe las fuerzas totales de las que va a disponer a lo largo de la competición, $F \in \mathbb{N}$, y quiere repartirlas entre las distintas pruebas para ganar la mayor cantidad posible de dinero. Se pide:

a) Formaliza el conjunto de soluciones factibles e indica, a partir de él, cómo se calcularía el valor de la función objetivo sobre una solución factible (esto no es preciso formalizarlo).

b) Si el nadador prepara 5 pruebas y $e = [40, 100, 30, 50, 20]$ y $F = 200$, indica, utilizando la notación definida en el apartado anterior, cuál sería una solución óptima sobre el ejemplo propuesto y su valor.

c) Propón una estrategia voraz lo más eficiente posible que devuelva la solución óptima para el problema ante cualquier entrada válida.

d) Diseña un algoritmo que siga dicha estrategia. El algoritmo deberá devolver, para cada prueba, el *esfuerzo* que debe realizar el nadador para alcanzar la solución óptima.

e) Indica (y justifica) cuál es el coste temporal de la estrategia (y del algoritmo) propuestos.

f) Supón que los premios por quedar en primer, segundo y tercer lugar fueran un dato de entrada al algoritmo: $p = (p_1, p_2, p_3) \in \mathbb{N}^3$, donde $p_1 > p_2 > p_3$ (por ejemplo, $p =$ (100.000, 75.000, 50.000)). ¿Seguiría siendo válida la estrategia que has diseñado? Justifica la respuesta. Si no es válida, ¿podrías modificarla o diseñar otra que garantizará encontrar la solución óptima para cualquier instancia del problema? Si la respuesta es positiva, indica cómo, y si es negativa, justifícala.

2. Divide y vencerás — 5 puntos

Disponemos de un vector v que contiene números enteros. Los elementos del vector se encuentran divididos en dos grupos de valores crecientes consecutivos, el primero de los grupos formado por números negativos y el segundo por números positivos. Sin embargo, no disponemos de información acerca de cuáles pueden ser esos valores ni la cantidad de valores que contiene cada grupo (aunque, al menos, de cada grupo hay un valor).

El vector $v = [-12, -11, -10. - 9, -8, -7, 9, 10, 11, 12, 13, 14, 15, 16]$ cumple dichas propiedades, donde el primer grupo empieza en la posición 0 y acaba en la 5 (del -12 al -7), y el segundo grupo empieza en la 6 y acaba en la 13 (del 9 al 16).

Queremos conocer la suma de los valores del primer grupo y la de los valores del segundo grupo, sabiendo que en un rango consecutivo de n valores, en el que v_i es el primer elemento del rango y v_j el último elemento, se cumple que la suma es $(v_i + v_j) * n/2$, donde $n = j - i + 1$. En el ejemplo, la suma del primer grupo es $(-12 + -7) * (5 - 0 + 1)/2 = -19 * 6/2 = -57$, mientras que la suma del segundo grupo es $(9 + 16) * (13 - 6 + 1)/2 = 100$. Se pide:

a) Diseña un algoritmo recursivo mediante la técnica de divide y vencerás que devuelva como resultado el par de valores correspondientes a la suma del primer grupo y a la suma del segundo grupo (para el ejemplo, debería devolver $(-57, 100)$). El algoritmo debe ser lo más eficiente posible tanto desde el punto de vista de la complejidad temporal como de la espacial.

   Una pista para obtener la versión más eficiente: *la función recursiva* que desarrolles 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 *mediante otra función*, obtener los valores pedidos. 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.

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) ¿Se te ocurre algún algoritmo iterativo sencillo que resuelva el mismo problema sin utilizar la técnica de divide y vencerás (no es necesario que lo implementes, tan sólo descríbelo)? Indica el coste asintótico que presenta y si, atendiendo a dicho coste, puede considerarse mejor o peor que el que has diseñado.

Segundo control parcial de Esquemas Algorítmicos (IG24)

27 de mayo de 2006

1. Programación dinámica — 4 puntos

En la preparación de una maratón por etapas se ha diseñado ya el trayecto concreto por el que han de correr los atletas, pero falta decidir dónde empieza y dónde acaba cada una de las E etapas de la maratón. El trayecto completo parte del kilómetro 0 y finaliza en el kilómetro K. Las etapas sólo pueden empezar o acabar en puntos kilómetricos enteros (por ejemplo, en el kilómetro 5, pero no en el 5.3).

El esfuerzo realizado por un corredor en una etapa depende de sus kilómetros de inicio y finalización, así como del número de etapas que ya ha cubierto. Con $f(i, j, e)$ cuantificamos el esfuerzo que supone recorrer los kilómetros del i al j en la etapa e. Hay una limitación en el diseño de las etapas: una etapa no puede medir menos de k_1 kilómetros ni más de k_2.

Debes averiguar, empleando la técnica de programación dinámica, dónde debe empezar y acabar cada una de las E etapas para garantizar que el recorrido suponga el *mayor* esfuerzo posible a los corredores.

1. Con las condiciones anteriormente expuestas:

   a) Formaliza el problema en términos de optimización (conjunto de soluciones factibles y función objetivo).

   b) Plantea la ecuación recursiva de programación dinámica que calcula el esfuerzo que requiere la maratón más costosa de realizar, indicando cuál debe ser la llamada que se efectue a dicha función para obtener el valor óptimo.

   c) Representa el grafo de dependencias entre las llamadas del algoritmo recursivo para el siguiente ejemplo de mentirijillas: $E = 3$ etapas, $K = 8$, $k_1 = 2$ y $k_2 = 5$. *Razona* los costes espacial y temporal con los que se puede efectuar el cálculo sobre dicho grafo (indica explícitamente, justificando tu respuesta, si es posible efectuar una reducción de complejidad espacial al realizar el cálculo iterativo).

   d) Si sabemos que la diferencia entre el máximo y el mínimo número de kilometros que es posible realizar en una etapa es $D = k_2 - k_1$, y que $D < K$, ¿podrías expresar de forma más ajustada el coste temporal del algoritmo? Justifica tu respuesta.

   e) ¿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 y pon algún ejemplo de instancia que no tenga solución factible.

2. El año siguiente, la organización de la carrera decide cambiar parte de los criterios para el diseño de la carrera: dada la duración total de la carrera K y la duración mínima y máxima de cada etapa, k_1 y k_2, desea realizar el mejor diseño en etapas, pero *sin fijar previamente el número total de etapas de la carrera.*

   ¿Cómo modificarías el conjunto de soluciones factibles para que recogiera este nuevo planteamiento? ¿Cómo resolverías ahora el problema? ¿Con qué coste temporal y espacial?

2. Programación dinámica — 2.75 puntos

Un vigilante nocturno de un polígono industrial tiene que fichar cada noche R veces en distintos puntos del polígono para demostrar que se dedica a su trabajo. En total hay N naves industriales (por simplicidad, asumiremos que están numeradas entre la 1 y la N) y en cada una de ellas hay un punto de fichaje.

El vigilante debe comenzar y acabar la ronda en un mismo punto de control, situado en la nave principal, cuyo número es el p. Para asegurarse de que no se instala a pasar la noche en una nave, no esta permitido que el vigilante fiche dos veces seguidas en un mismo puesto, aunque que no hay inconveniente en que el vigilante fiche más de una vez en un mismo punto durante la noche siempre que respete la regla anterior.

El vigilante, que sabe la distancia que hay entre cada par de naves ($d(n', n)$, donde $1 \leq n' \leq N, 1 \leq n \leq N, n' \neq n$), desea que le indiques en que naves debe fichar para recorrer la menor distancia a lo largo de una jornada. Como ayuda, la siguiente ecuación recursiva de programación dinámica permite resolver el problema de obtener el valor de la mínima distancia:

$$D(r, n) = \begin{cases} 0, & \text{si } r = 1 \text{ y } n = p; \\ +\infty, & \text{si } r = 1 \text{ y } n \neq p; \\ \min\limits_{\substack{1 \leq n' \leq N: \\ n' \neq n}} D(r-1, n') + d(n', n), & \text{si } r > 1. \end{cases}$$

La llamada $D(R, p)$ proporciona dicho valor.

Se pide:

a) Representa el grafo de dependencias para el problema en el que $R = 4$, $N = 5$ y $p = 2$. Indica qué representa un estado (r, n) y un arco entre dos estados (r', n') y (r, n) dentro del grafo.

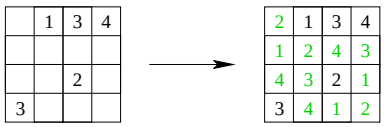
b) Diseña un algoritmo iterativo que devuelva la *secuencia de naves óptima* en las que debe hacer los fichajes para obtener la mínima distancia.

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 de la mínima distancia? Si es así, índica cuál sería dicho coste y justifícalo.

3. Búsqueda con retroceso — 3.25 puntos

El simpleku es una variante del juego del sudoku que consiste en ubicar en un tablero de n filas por n columnas los números del 1 al n de forma que en todas las filas del tablero y en todas las columnas aparezcan los n números (o, lo que es lo mismo, un mismo número aparecerá una vez en cada columna y una vez en cada fila). Para darle sentido al juego, se falicita un tablero en el que ya aparecen ubicados algunos de los números, teniendo que encargarse el jugador de rellenar las restantes casillas. Por ejemplo, para un problema con $n = 4$, la siguiente figura muestra una configuración inicial y una posible solución:



Queremos encontrar un algoritmo de *búsqueda con retroceso* que rellene un tablero de $n \times n$ a partir de una configuración inicial. El tablero con la configuración inicial se facilita en forma de matriz, estando rellenas con el valor 0 las casillas inicialmente vacias. También se proporciona una lista con las posiciones libres, en las que cada elemento de la lista es un par de valores con las coordenadas de la posición en el tablero. En el ejemplo anterior, los datos de entrada del problema serían:

```
tab = [[0,1,3,4],
       [0,0,0,0],
       [0,0,2,0],
       [3,0,0,0]]
libres = [(0,0),(1,0),(1,1),(1,2),(1,3),(2,0),(2,1),(2,3),(3,1),(3,2),(3,3)]
```

A continuación, se presenta el esquema de un posible algoritmo de *backtracking* que resuelve este problema:

```
def simpleku(n,tab,libres):
  def backtracking(pos):
    if is_complete(pos):
      if is_factible(pos): return tab
    else:
      i,j=libres[pos]
      for num in xrange(1, n+1):
        if is_promising(i,j,num):
          tab[i][j]=num
          found = backtracking(pos+1)
          if found != None: return found
      tab[i][j]=0
    return None
  return backtracking(0)
```

Se pide que:

a) Escribas el código correspondiente a la función `is_complete(pos)`, es decir, que definas cuál es el criterio que permite determinar si un estado es una *solución completa*.

b) Escribas el código correspondiente a la función `is_factible(pos)`, es decir, que definas cuál es el criterio que permite determinar si una *solución completa* es una *solución factible* o no.

c) Escribas el código correspondiente a la función `is_promising(i,j,num)`, es decir, que definas cuál es el criterio que permite determinar si un *estado parcial* puede conducir a una solución factible (en cuyo caso hay que ramificarlo) o no.

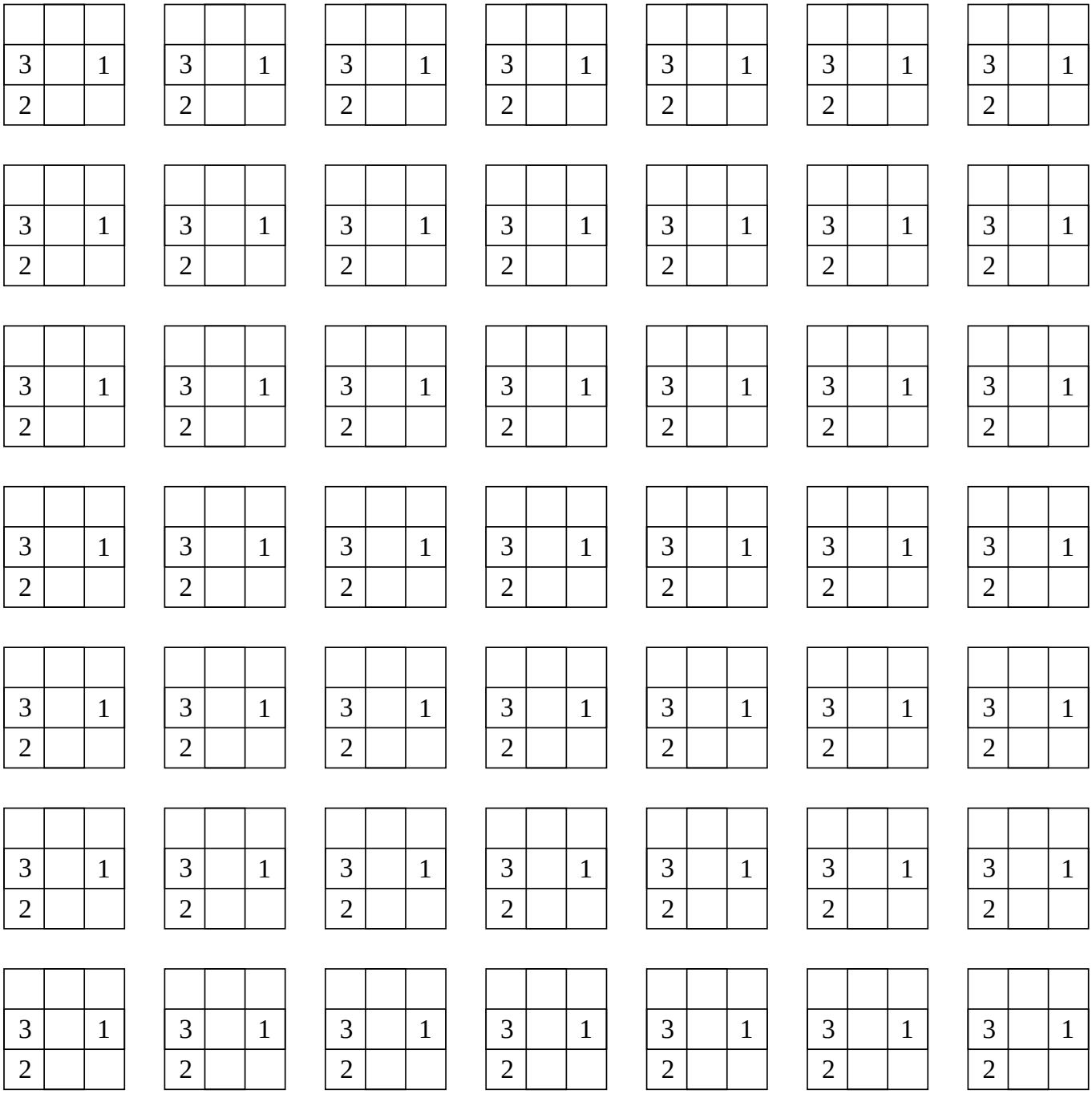
d) A la vista del código desarrollado en los apartados anteriores, ¿cuál sería la complejidad espacial del algoritmo desarrollado? ¿Y la complejidad temporal en el *mejor de los casos*? Justifícalas.

e) Realiza una traza de ejecución de este algoritmo para la llamada `simpleku(n,tab,libres)` donde $n = 3$ y

```
tab = [[0,0,0],
       [3,0,1],
       [2,0,0]]
libres = [(0,0),(0,1),(0,2),(1,1),(2,1),(2,2)]
```

Para ello, haz uso de las figuras que aparecen en la siguiente hoja (sólo de aquellas que necesites). 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.

f) Por último, modifica lo que consideres oportuno del algortimo para introducir una restricción adicional: todos los números deben aparecer también en las diagonales principales del tablero (es decir, un número no puede aparecer más de una vez en cada diagonal principal).


3 1
2

Examen convocatoria ordinaria de Esquemas Algorítmicos (IG24)

16 de junio de 2006

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) ( ) Todo ( ) 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. Voraces — 2,5 puntos

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

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

a) Los símbolos del alfabeto se nos proporcionan inicialmente ordenados de menor a mayor frecuencia de aparición.

b) Los símbolos del alfabeto presentan todos la misma frecuencia de aparición.

c) Los símbolos del alfabeto cumplen que hay uno de ellos que se presenta la mitad de las veces (su frecuencia es 1/2), otro se presenta un cuarto de las veces (1/4), otro 1/8, otro 1/16 y así sucesivamente. Los símbolos se nos facilitan desordenados.

d) El mismo caso que el anterior, pero los símbolos se nos proporcionan ordenados de mayor a menor frecuencia.

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

2. Divide y vencerás — 2,5 puntos

Dado un vector (sin ordenar) compuesto por n números enteros, X, y dos números a y m, se desea saber si el número a aparece más de m veces en el vector X.

a) Diseña un algoritmo recursivo mediante la técnica de divide y vencerás que permita resolver este problema (con una *complejidad espacial no superior* a $O(\log n)$). Recuerda que debes especificar la función que recibe los parámetros y realiza la llamada inicial que desencadena el procedimiento recursivo que efectúa los cálculos necesarios (no olvides indicar igualmente esta llamada).

   Pista: para facilitar la elaboración del algoritmo, puedes hacer que el procedimiento recursivo se dedique a contar el número de veces que aparece a en el (trozo de) vector correspondiente y que la función que recibe los parámetros y realiza la llamada inicial sea la que realmente devuelva el resultado `True` o `False` requerido.

b) Indica si el algoritmo que has diseñado en el apartado anterior presenta un comportamiento uniforme o existe un mejor y un peor caso. Si el comportamiento es uniforme, modifícalo para que tenga un mejor y un peor caso. Finalmente, señala en qué condiciones (ante qué tipo de entradas) puede el algoritmo comportarse en el mejor de los casos y en qué condiciones en el peor.

c) Analiza la *complejidad temporal* del algoritmo resultante tras la realización del apartado anterior. Debes justificar adecuadamente los costes calculados.

d) ¿Se te ocurre algún algoritmo iterativo sencillo que resuelva el mismo problema sin utilizar la técnica de divide y vencerás (no es necesario que lo implementes, tan sólo descríbelo)? Indica el coste asintótico que presenta y si, atendiendo a dicho coste, puede considerarse mejor o peor que el que has diseñado.

3. Programación dinámica — 3,5 puntos

Una cadena de televisión dedica una franja *consecutiva* de su horario de madrugada a la publicidad, con la emisión de publireportajes y anuncios. La duración de dicha franja es de M minutos. Para cubrirla, dispone de N anuncios y publis, numerados de 1 a N, de los que conoce su duración en *segundos*, $s_1, s_2, \ldots, s_N$. Asimismo, sabe cuál es el precio, $p_i, 1 \leq i \leq N$, que están dispuestas a pagar las compañías anunciantes por la emisión de sus anuncios.

La cadena desea averiguar cuál es el máximo beneficio que puede obtener con la emisión de cualquier combinación de anuncios. Para ello, la cadena asume que no tiene por qué emitir todos los anuncios disponibles y que en la franja de publicidad puede repetir la emisión de un anuncio tantas veces como le interese, teniendo, eso sí, que ocupar completamente dicha franja con los anuncios disponibles. Resuelve el problema de la forma más eficiente posible utilizando la técnica de programación dinámica, para lo que se te pide:

a) Formaliza el problema en términos de optimización,

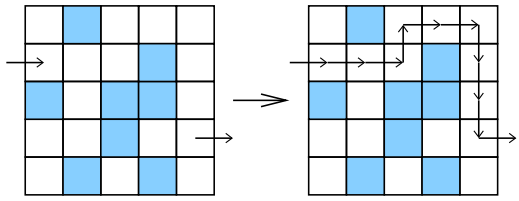
b) plantea la ecuación recursiva que calcula el máximo beneficio posible,

c) diseña un algoritmo iterativo que devuelva dicho valor, razonando su coste temporal y espacial (indica explícitamente, justificando tu respuesta, si es posible efectuar una reducción de complejidad espacial al realizar el cálculo iterativo), y

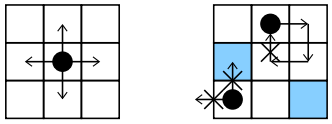
d) razona si puede haber instancias del problema para las que no exista una solución factible.

4. Búsqueda con retroceso — 1,5 puntos

Queremos encontrar un algoritmo de *búsqueda con retroceso* que, partiendo de una casilla inicial (pos_ini= (i, j)), atraviese un laberinto representado mediante un tablero de $n \times n$ y alcance una casilla final (pos_fin= (i', j')).



(a)



(b)

Figura 1: (a) Ejemplo de laberinto y recorrido a traves del laberinto. (b) Izquierda: desplazamientos posibles desde una casilla. Derecha: desplazamientos y caminos no permitidos.

El laberinto, que se facilita como una matriz booleana, está formado por casillas libres (valor `True`), a través de las que se puede transitar, y por casillas bloquedas (valor `False`), a las que no se puede acceder. En una trayectoria a traves del laberinto se pasa de una casilla libre a otra. Para ello hay que tener en cuenta que, dada una casilla libre (i, j), es posible desplazarse directamente desde ella a una de las cuatro casillas adyacentes (en horizontal o vertical, casillas $(i+1, j)$, $(i, j+1)$, $(i-1, j)$ e $(i, j-1)$). La casilla a la que nos desplacemos tiene que ser una casilla libre. Además, hemos de tener cuidado de no salirnos del tablero y, para evitar estar dando vueltas sin fin en el laberinto, no está permitido pasar más de un vez por una misma casilla (en el camino finalmente elegido desde la posición inicial a la final).

En el ejemplo de la figura (a), los datos de entrada del problema serían:

```
n=5; pos_ini=(1,0); pos_fin=(3,4)
lab=[[True,False,True,True,True],
     [True,True,True,False,True],
     [False,True,False,False,True],
     [True,True,False,True,True],
     [True,False,True,False,True]]
```

A continuación, se presenta el esquema de un posible algoritmo de búsqueda con retroceso que resuelve este problema:

```
def laberinto(n,lab,pos_ini,pos_fin):
  sol = [pos_ini]
  def backtracking(ult_pos):
    if is_complete(ult_pos):
      return sol
    else:
      (i,j)=ult_pos
      for pos in [(i+1,j),(i,j+1),(i-1,j),(i,j-1)]:
        if is_promising(pos):
          sol.append(pos)
          found = backtracking(pos)
          if found != None: return found
          sol.pop()
      return None
  return backtracking(pos_ini)
```

Si llamamos a la función `laberinto` con los datos del ejemplo anterior: `laberinto(5,lab,(1,0),(3,4))`, el resultado será:

```
[(1, 0), (1, 1), (1, 2), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4)]
```

que se corresponde con la solución mostrada en la figura (a). Se pide que:

a) Escribas el código correspondiente a la función `is_complete(ult_pos)`, es decir, que definas cuál es el criterio que permite determinar si un estado es una *solución completa* (y, en este caso, también factible).

b) Escribas el código correspondiente a la función `is_promising(pos)`, es decir, que definas cuál es el criterio que permite determinar si un *estado parcial* puede conducir a una solución factible o no.

c) A la vista del código desarrollado en los apartados anteriores, ¿cuál sería la complejidad temporal en el *mejor de los casos* del algoritmo desarrollado? Justifícala. ¿Crees posible reducir dicho coste? Indíca cómo.

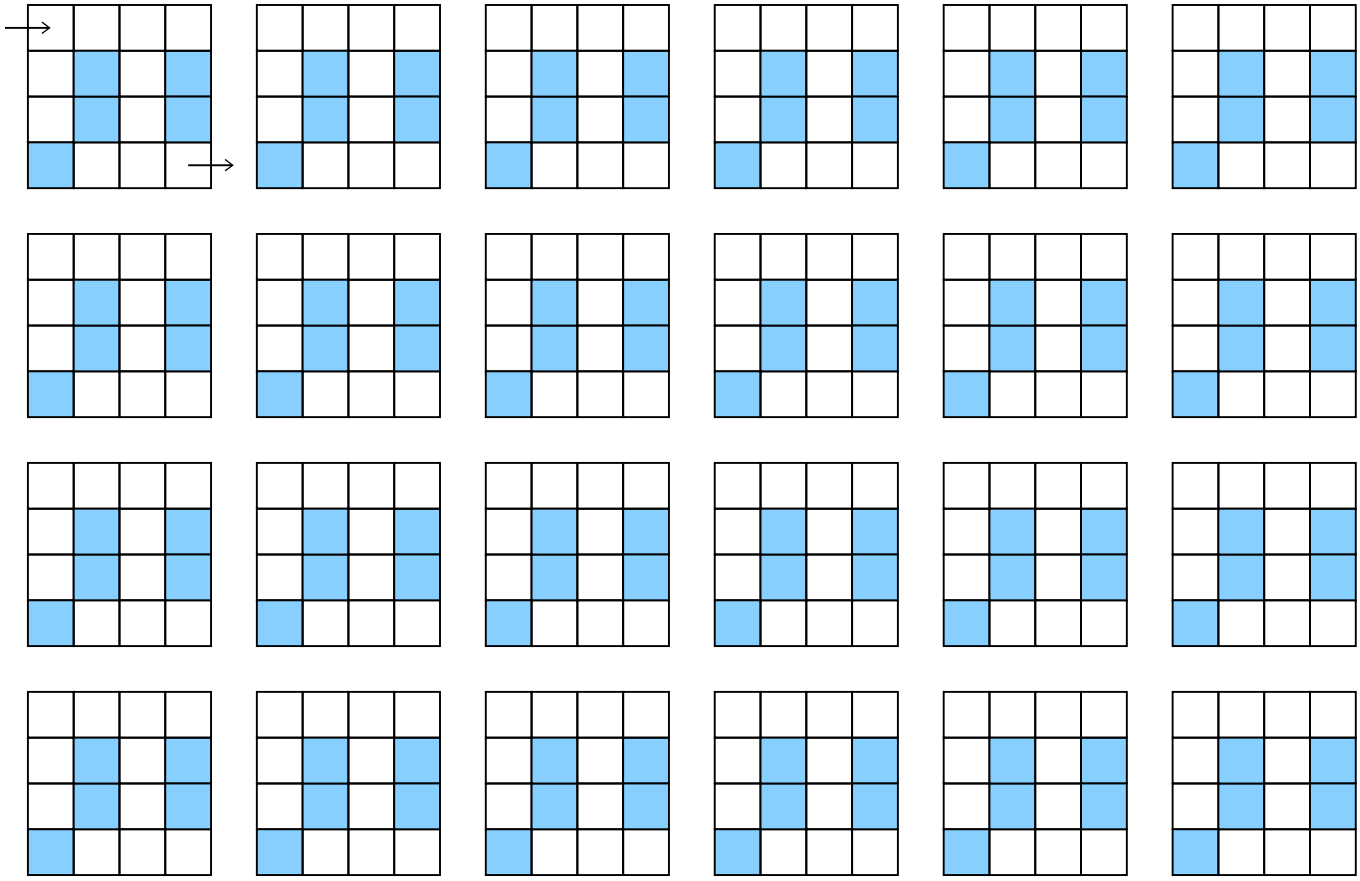
d) Realiza una traza de ejecución de este algoritmo para la llamada

```
laberinto(4,lab,(0,0),(3,3))
```

donde

```
lab=[[True,True,True,True],
     [True,False,True, False],
     [True,False,True, False],
     [False,True,True,True]]
```

Para ello, haz uso de las figuras que aparecen a continuación (sólo de aquellas que necesites). En cada una de ellas debes indicar con claridad la casilla que está estudiando el algoritmo, 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.



e) Por último, modifica el algoritmo para que, en lugar de devolver un camino a través del laberinto devuelva el número de caminos distintos que permiten atravesar el laberinto desde la casilla de salida a la de llegada.

Examen segunda convocatoria de Esquemas Algorítmicos (IG24)

5 de septiembre de 2006

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) ( ) Todo ( ) 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. Voraces — 2,5 puntos

En el almacén de una cafetería hay n cajas con helados de m diversos sabores. Cada caja contiene helados de un único sabor (por supuesto, puede haber varias cajas con helados del mismo sabor). Los propietarios han decidido utilizar para los postres del menú diario de la cafetería una única caja de helados completa (una al día) a partir del 1 de junio y hasta que se acaben todas las cajas (bien porque se hayan consumido, bien porque hayan caducado). Tenemos dos vectores indexados por el número de caja c ($1 \leq c \leq n$): $D[c]$ indica el número de días que faltan para que caduquen los helados de la caja c a partir del 1 de junio, mientras que $S[c]$ indica el sabor de los helados que hay en dicha caja. Además, la cafetería maneja un estudio estadístico que muestra la aceptación de los diversos sabores de los helados; así, tenemos otro vector, A, tal que para cada sabor i ($1 \leq i \leq m$), $A[i]$ indica el número de clientes a los que les gustan los helados de dicho sabor.

Un ejemplo con $n = 10$ cajas y $m = 3$ diferentes sabores podría tener como días de caducidad para las cajas de la 1 a la 10 $D = [10, 5, 1, 20, 8, 1, 5, 5, 3, 10]$, cuyos números de sabores serían $S = [1, 2, 1, 3, 3, 1, 2, 3, 2, 2]$ y donde las respectivas aceptaciones de los tres sabores fueran $A = [50, 30, 40]$.

Por último, cabe tener en cuenta la siguiente restricción adicional: existe una función, `compatible(i,j)` (*no simétrica*), que devuelve `True` si podemos poner de postre un helado de sabor j dado que el día anterior pusimos uno de sabor i y `False` en caso contrario ($1 \leq i \leq m, 1 \leq j \leq m$). Los propietarios de la cafetería nos piden que desarrollemos una aplicación informática que determine la secuencia de cajas de helados que hay que servir en el menú diario, de manera que se maximice el número de clientes a los que les gustan los helados servidos en los diversos menús. Hemos pensado desarrollar una solución mediante un algoritmo *voraz*, pero necesitamos tu ayuda. Se pide que:

a) Ya que se trata de un problema de optimización, indiques las características que ha de poseer una *solución factible*, cuál es la función objetivo que se pretende optimizar y qué propiedad debe cumplir una solución factible para ser considerada óptima.

b) Considera la siguiente estrategia voraz:

   "Ordenamos las cajas de menor a mayor caducidad en una lista C y mientras queden cajas en C:

   - Extraemos la primera caja de la lista,
   - si la caja ha caducado, se descarta,
   - si no ha caducado pero no es compatible con la anterior, se añade al final de una segunda lista C',
   - si la caja no ha caducado y es compatible con la anterior, se añade la caja a la solución y se incrementa en 1 el número de días transcurridos (inicialmente fijado a 0).

   A continuación, y si hay cajas en la segunda lista C', esta se recorre igual que la primera, sólo que en este caso, si la caja extraida ha caducado *o no es compatible* se descarta y, si no, se añade a la solución incrementando también en 1 el número de días."

   Con respecto a esta estrategia:

   1. ¿Garantiza encontrar una solución factible, si es que ésta existe? Justifica tu respuesta (si es negativa, debes dar un contraejemplo).
   2. ¿Encuentra siempre una solución óptima? Justifica la respuesta (si es negativa, debes dar un contraejemplo).
   3. ¿Cuál sería la complejidad temporal y espacial de esta estrategia? Justifícalas.
   4. ¿Es posible que se puedan desechar cajas que no están caducadas y podrían aprovecharse a la hora de servir el postre en algún menú? Si es así, pon un ejemplo.

c) Planteamos otra estrategia voraz similar a la anterior, excepto en que inicialmente lo que hacemos es ordenar las n cajas de mayor a menor aceptación de su sabor, teniendo en cuenta que las cajas de idéntico sabor se ordenan entre sí de menor a mayor caducidad. Responde a las mismas preguntas (1, 2, 3 y 4) del anterior apartado para esta nueva estrategia.

2. Divide y vencerás — 2,5 puntos

La *mediana* M de un conjunto de valores es aquel elemento que tiene tantos elementos menores como mayores en el conjunto (es el valor que, si el conjunto estuviera ordenado, ocuparía la posición central). Sean v y w dos vectores *ordenados* de manera *no decreciente* con n números enteros cada uno ($n > 0$). Se desea implementar un algoritmo que obtenga eficientemente la mediana M de los $2n$ elementos que totalizan ambos vectores.

Para ello, date cuenta que podemos calcular de forma inmediata las medianas de cada vector, M_v y M_w, ya que ambos se encuentran ordenados. Si ocurre que $M_v = M_w$ entonces M coincide con ellas. Si $M_v \neq M_w$, entonces puede ocurrir que $M_v < M_w$ o bien que $M_v > M_w$. En el primer caso, podemos afirmar que M va a ser mayor que M_v pero menor que M_w, mientras que en el segundo sabemos que M va a ser mayor que M_w pero menor que M_v. Fíjate también en que si $n = 1$, entonces la mediana sería el *mínimo* entre los dos números posibles, el de v y el de w.

a) Conociendo este hecho, has de diseñar un algoritmo eficiente basado en la técnica de *divide y vencerás* para obtener M. Recuerda que debes especificar la función que recibe los parámetros y realiza la llamada inicial que desencadena el procedimiento recursivo que efectúa los cálculos necesarios (no olvides indicar igualmente esta llamada).

b) Analiza y justifica la *complejidad temporal* y *espacial* del algoritmo desarrollado en el apartado anterior *en función de* n, indicando si presenta un comportamiento uniforme o existe un mejor y un peor caso. Si el comportamiento es uniforme, modifícalo para que tenga un mejor y un peor caso. Finalmente, señala en qué condiciones (ante qué tipo de entradas) puede el algoritmo comportarse en el mejor de los casos y en qué condiciones en el peor.

c) Propón un algoritmo iterativo que resuelva este mismo problema *sin* utilizar la técnica de divide y vencerás. Indica el coste asintótico que presenta y si, atendiendo a dicho coste, puede considerarse mejor o peor que el resultante tras la realización del apartado b).

3. Programación dinámica — 3,5 puntos

Una cadena de televisión dispone de una serie de n anuncios publicitarios, cada uno con una duración concreta (en segundos) $(d_1, d_2, \ldots, d_{n-1}, d_n)$, que deben emitirse *exactamente en ese orden* en el horario de *prime time*. Para emitir los anuncios, los responsables de la cadena pueden asignar tantas franjas como sean necesarias de D segundos (el tope máximo establecido por ley, obviamente se asume $\forall i$ que $d_i \leq D$). Así pues, en cada franja se emitirá una tanda de anuncios[1], cuya duración nunca puede exceder de D segundos; aunque puede sobrar tiempo. Los responsables de la cadena estiman que si en una franja se pierden p segundos, las perdidas económicas asociadas a la franja son de p^2 €.

La cadena desea averiguar la cantidad de franjas necesarias y cuáles serían los bloques de anuncios que debe emitir en cada una de ellas de tal manera que las *pérdidas sufridas fuesen mínimas*.

Veamos un ejemplo: tenemos 8 anuncios con duraciones, respectivamente, $d_1 = 20$ s., $d_2 = 30$ s., $d_3 = 10$ s., $d_4 = 15$ s., $d_5 = 50$ s., $d_6 = 45$ s., $d_7 = 35$ s. y $d_8 = 20$ s; siendo $D = 75$ s. Una posible solución consistiría en programar 4 franjas: la primera emitiría los anuncios del 1 al 4 ($\texttt{segs}(1,4) = 75 \rightarrow p = 0$), la segunda el 5 ($\texttt{segs}(5,5) = 50 \rightarrow p = 25$), la tercera el 6 ($\texttt{segs}(6,6) = 45 \rightarrow p = 30$) y la cuarta los anuncios 7 y 8 ($\texttt{segs}(7,8) = 55 \rightarrow p = 20$). Las pérdidas serían $0^2 + 25^2 + 30^2 + 20^2 = 1925$ €. Fíjate que no sería posible emitir los anuncios 5 y 6 en una misma franja horaria ya que ambos totalizan 95 segundos; cantidad superior a los 75 que constituye el límite legal.

Sin embargo, existe otra solución mejor, también usando 4 franjas: en la primera se emiten los anuncios del 1 al 3 ($\texttt{segs}(1,3) = 60 \rightarrow p = 15$), en la segunda se emiten los anuncios 4 y 5 ($\texttt{segs}(4,5) = 65 \rightarrow p = 10$), en la tercera se emite el 6 ($\texttt{segs}(6,6) = 45 \rightarrow p = 30$) y en la cuarta el 7 y 8 ($\texttt{segs}(7,8) = 55 \rightarrow p = 20$). En este caso, las pérdidas serían de $15^2 + 10^2 + 30^2 + 20^2 = 1625$ €. Tienes que resolver este problema de la forma más eficiente posible utilizando la técnica de *programación dinámica*, para lo que se te pide:

a) Formaliza el problema en términos de optimización.

b) Plantea la ecuación recursiva que calcula la mínima pérdida posible. Tampoco olvides indicar cómo se calcula la solución óptima (es decir, qué llamada o llamadas iniciales deben efectuarse).

c) Diseña un algoritmo un algoritmo recursivo *con memorización* que resuelva eficientemente la recurrencia planteada en el apartado anterior.

d) Representa el grafo de dependencias entre estados a partir de la recurrencia desarrollada en el apartado b) para el ejemplo propuesto.

e) A partir del grafo anterior, indica y justifica la complejidad temporal y espacial del algoritmo *iterativo* de programación dinámica que resuelve el problema de calcular *la asignación de anuncios a franjas* cuya emisión produce las pérdidas mínimas.

4. Búsqueda con retroceso — 1,5 puntos

Una empresa de desarrollo de software para terceros tiene en perspectiva la realización de n aplicaciones diferentes, cada una solicitada por un cliente distinto. En la empresa trabajan un total de m analistas ($m > n$). La empresa desea asignar un analista *distinto* a cada uno de los proyectos. Para ello, se debe tener en cuenta que no todos los analistas tienen los conocimientos adecuados para desarrollar satisfactoriamente la aplicación. En este sentido, puedes asumir que se dispone de una función *booleana*, `domina(a,p)`, que devuelve `True` si el analista a tiene los conocimientos requeridos para llevar a cabo el proyecto p y `False` en caso contrario.

Además, puesto que ya se han tenido anteriores experiencias profesionales con los clientes y se han producido ciertos "roces" con el personal de la empresa de software, no se desea asignar un analista que haya tenido problemas con el cliente al proyecto que ha solicitado. Nuevamente, asume que existe una función *booleana*, `adecuado(a,p)`, que devuelve `True` si resulta apropiado asignar el analista a al proyecto p y `False` en caso contrario. Hay que desarrollar una solución mediante la técnica de *búsqueda con retroceso* que permita encontrar, si es que existe, la asignación de analistas a proyectos de manera que los analistas asignados conozcan las técnicas necesarias y no hayan tenido problemas anteriormente con el cliente solicitante. Para ello:

[1] Para facilitar los cálculos, se dispone de la función `segs` que, dados los números del primer y último anuncio de una franja, calcula los segundos invertidos en la franja: `segs`(i,j)$= d_i + d_{i+1} + \ldots + d_j = \sum_{i \leq k \leq j} d_k$.

a) Describe cómo vas a representar los estados en la resolución del problema planteado (en qué van a consistir las tuplas, cuantos elementos van a contener, valor posible de esos elementos).

b) Atendiendo a la representación escogida en el apartado anterior, ¿cuál sería la función de *ramificación* (`branch`)? Debes indicar el código o bien una descripción lo *más precisa* posible.

c) Describe la implementación de la función que permite determinar si una solución que se acaba de ramificar es prometedora o no (`is_promising`): igualmente se pide el código o bien una descripción lo *más precisa* posible.

d) Teniendo en cuenta las decisiones tomadas en los apartados anteriores, hemos de depurar el código haciendo *una traza* con los siguientes datos: se dispone de 6 analistas, que deseamos asignar a 4 proyectos distintos, y las funciones `domina(a,p)` y `adecuado(a,p)` devuelven `False` en los proyectos indicados en la siguiente tabla:

analista	`domina`	`adecuado`
1	{1,3}	-
2	{2}	{1}
3	{4}	{2}
4	{1,4}	{4}
5	{3}	{4}
6	{4}	-

Recuerda que debes dibujar el árbol de llamadas incluyendo sólo aquellos estados que se generan durante el proceso de búsqueda y marcando de forma distinta aquellas soluciones que no se ramifican por no ser prometedoras. Indica cuál es el resultado devuelto por el algoritmo.