Escuela de Ingeniería Informática
Universidad de Oviedo

Algoritmia
2014-2015

Parcial 2 – 13 de abril de 2015

Apellidos, nombre _____ NIF: _____

Pregunta 1 (4 puntos)

Los mundiales de natación de Barcelona 2013 han sido un gran paso para la natación española con 12 medallas en total. Una de las pruebas que se ha decidido mejorar para el próximo mundial es el relevo 4x100 estilos femeninos, en el que participan 4 nadadoras cada una en un estilo de natación diferente: espalda (E), braza (B), mariposa (M) y crol (C). El seleccionador dispone de cuatro nadadoras y de sus tiempos para cada una de las pruebas. Se quiere diseñar el algoritmo que mediante la técnica de backtracking permita decidir al seleccionador que nadadora participará en cada estilo de tal forma que optimicen los tiempos del conjunto.

a) (3 puntos) Completar el código Java para dar solución a este problema (escribirlo en los recuadros preparados a tal efecto).

```
static int nEstilos;
static int[][]tiemposNadadoras;  // tiempos de las nadadoras
static boolean [] marca;

static int[]seleccion;
static int tiempo;// su tiempo

static int[]seleccionMejor;// para anotar la mejor
static int tiempoMejor;// tiempo de la mejor


static void backtracking (int nadadora)
{
      if (nadadora==nEstilos)
      {
            if (tiempo<tiempoMejor)
            {
                  for (int k=0;k<nEstilos;k++)
                        seleccionMejor[k]=seleccion[k];
                  tiempoMejor=tiempo;
            }
      }
      else
      {
            for (int est=0; est<nEstilos; est++)
                  if (!marca[est])
                  {
seleccion[nadadora]=est;
tiempo+= tiemposNadadoras[nadadora][est];
marca[est]=true;
backtracking (nadadora+1);
tiempo-= tiemposNadadoras[nadadora][est];
marca[est]=false;
                  }
      }
}
```

a) (1 punto) Razonar si esta técnica es eficiente para realizar el cálculo y qué otras técnicas vistas en clase podrían dar solución a este problema indicando las ventajas y los inconvenientes.

La técnica de backtracking nos proporciona solución óptima, pero con el coste de O(n!)

Los algoritmos voraces que tienen una complejidad O(n^2) para este caso, con lo cual son mucho más eficientes, pero no dan soluciones óptimas en todos los casos, habría que afinar en los requisitos del problema si es imprescindible que siempre tengamos la solución óptima para poder emplearla.

Para emplear otras técnicas como Divide y vencerás o programación dinámica, en primer lugar deberíamos disponer de una función recursiva de la cual no disponemos, por tanto, no podemos plantear una solución por estas técncias.

Pregunta 2 (3 puntos)

El algoritmo de Floyd consiste en encontrar el camino más corto entre todos los pares de nodos de un grafo. Dado un grafo G, lo que queremos es hallar el camino más corto para ir desde un nodo i hasta otro nodo j de forma directa o a través de otro nodo k.

Para resolver este problema tenemos disponible la matriz de pesos del grafo (G(i,j)), es decir, una matriz que recoge el coste de ir de un nodo i a otro j del grafo, sin pasar por nodos intermedios. Cuando no hay camino directo entre dos nodos, la celda correspondiente tendrá valor infinito. Y se aplica la siguiente función D(i,j)= min(D(i,j), D(i,k)+D(k,j)) pudiendo ser k cualquier otro nodo del grafo.

a) Razonar por qué decimos que Floyd se encuadra dentro de las técnicas de programación dinámica.

La solución depende de una función recursiva *D(i,j)*; pero en vez de darle una solución implementándola recursivamente, representamos los datos en una tabla y vamos construyendo la solución sobre esta con un algoritmo iterativo, partiendo de los casos triviales hasta que llegamos al caso que nos piden solucionar.

b) Explicar que valores se pueden rellenar de forma directa en la tabla.

Floyd vuelca la matriz de pesos en la matriz D para comenzar el cálculo partiendo de los valores de los caminos directos del grafo.

c) Marcar sobre el array del apartado b) las casillas de las que depende (2,3) para calcular su valor.

Origen / destino	1	2	3	4
1			(1,3)	
2	(2,1)		X (2,3)	(2,4)
3				
4			(4,3)	

d) (1 punto) Rellenar la matriz D y la matriz de P de caminos que permite recuperar la secuencia de nodos para el camino más corto entre dos nodos con los siguientes datos de entrada:

Origen / destino	1	2	3	4
1	INF	3	4	2
2	INF	INF	INF	3
3	1	4	INF	INF
4	-1	INF	6	INF

La matriz D de costes mínimos es:

Origen / destino	1	2	3	4
1	**1**	**3**	**4**	**2**
2	**2**	**5**	**6**	**3**
3	**1**	**4**	**5**	**3**
4	**-1**	**2**	**3**	**1**

Realmente con los caminos existentes también se pueden crear caminos indirectos de un nodo a sí mismo, que en un principio no existían en el grafo.

Y la matriz P de predecesores es:

Origen / destino	1	2	3	4
1	**4**	**-1**	**-1**	**-1**
2	**4**	**4**	**4**	**-1**
3	**-1**	**-1**	**1**	**1**

4	**-1**	**1**	**1**	**1**

-1, indica que hay un camino directo, podría aparecer el índice correspondiente al nodo origen o destino.

e) Devolver el valor y la secuencia de nodos obtenida en el apartado anterior, cuando se va desde el nodo 2 al nodo 3.

El camino sería el siguiente: Nodo2 -> Nodo4 -> Nodo1 -> Nodo3. Y su coste 6.

Se puede utilizar el fichero Floyd.java proporcionado en seminarios para solucionar este problema y ver en detalle el funcionamiento de Floyd para construir las matrices y recuperar el camino óptimo.

Pregunta 3 (3 puntos)

Tenemos una colección de **n** ficheros que nos resultan de mucho valor, aunque lamentablemente necesitaremos borrar alguno de ellos porque nos hemos quedado sin espacio en nuestro disco duro de 600 MB.

Por ese motivo, tenemos que ingeniárnoslas para tratar que los ficheros con los que finalmente nos quedamos en el disco duro tengan el máximo valor posible para nosotros (los cuantificaremos en base a lo importantes que sean para nosotros). No podemos dividir los ficheros en partes más pequeñas porque dejarían de funcionar en nuestro ordenador, por lo que o nos quedamos con un fichero o lo descartamos.

a) Indica tres heurísticos que podíamos utilizar en un algoritmo voraz para calcular con qué ficheros nos quedamos.

Algunos heurísticos que podríamos utilizar son:

- Quedarnos con los ficheros en orden decreciente de importancia para nosotros.
- Quedarnos con los ficheros en orden creciente de tamaño.
- Quedarnos con los ficheros en orden decreciente cuyo valor por unidad de tamaño sea lo máximo posible.

b) Indica si dichos algoritmos son óptimos o no y demuéstralo con un contraejemplo.

Ninguno de los heurísticos es óptimo sin la posibilidad de dividir los ficheros. La demostración de que ni el primero ni el tercero son óptimos está a continuación:

Número de fichero	1	2	3
Tamaño	500 MB	300 MB	300 MB
Valor que le doy	1100	600	600
Heurístico 3	2.2	2	2

Con el heurístico 1 o con el heurístico 3 sólo cabría en el disco duro de 600 MB el fichero número 1. Sin embargo, está claro que la mejor solución sería utilizar los ficheros 2 y 3, que en total sumaría un valor de 4.

Por último, para demostrar que el heurístico 2 tampoco es óptimo basta con un sencillo ejemplo:

Número de fichero	1	2	3
Tamaño	100 MB	200 MB	600 MB
Valor que le doy	500	500	600000

Obviamente, la mejor opción en este caso sería utilizar únicamente el fichero número 3.