Universidad Rey Juan Carlos

Diseño y Análisis de Algoritmos - Grado en Ingeniería Informática

Prueba Final SOLUCIONES – Curso 2012/13

35 % de la nota final (3,5 puntos sobre 10)
13 de mayo de 2013. Duración: 2 horas
El código pedido deberá estar escrito en Java
Cada ejercicio se responderá en hojas separadas

Ejercicio 1 – El problema del hortelano [2 puntos]

Un hortelano posee n huertas, cada una con un tipo diferente de árbol frutal. Las frutas han madurado y es hora de recolectarlas. La recolección de una huerta dura un día completo. Se sabe, para cada huerta, el beneficio de lo que obtendría por la venta de lo recolectado. También se sabe los días que tardan en pudrirse los frutos de cada huerta. Este ejercicio está relacionado con **estrategias voraces** que ayuden al hortelano a decidir qué debe recolectar y cuándo debe hacerlo de forma que maximice el beneficio. Se pide:

a) Enumera y comenta muy brevemente la/s estrategia/s de selección voraces que se te ocurren para el problema **[0.5 puntos]**.

b) Escribe el programa que implemente la estrategia voraz **ÓPTIMA [1.5 puntos]**.

Al programa se le pasará una matriz $\mathbf{H}$ de tamaño $2 \times n$ con las fechas de caducidad (número de días que tardan en pudrirse los frutos) y con los beneficios de las n huertas. Siendo p el índice de una huerta (un entero de 0 a $n-1$), la fecha de caducidad de la huerta p es $\mathbf{H}[0][p]$ y su beneficio es $\mathbf{H}[1][p]$.

La salida es un vector $\mathbf{s}$ que contiene índices de las huertas de tal forma que $\mathbf{s}[m] = p$ significa que la huerta p se recolecta en el día $m+1$. Si un día no se recolecta el correspondiente valor en $\mathbf{s}$ será -1. Además asumimos que se conoce el valor de la caducidad máxima de las huertas (denotado por k).

A continuación se muestra un ejemplo de la salida óptima $\mathbf{s}$ para una determinada matriz $\mathbf{H}$, donde $n = 5$ y $k = 8$:

- Fechas de caducidad: $\mathbf{H}[0][0 : n-1] = [3, 8, 2, 3, 1]$
- Beneficios: $\mathbf{H}[1][0 : n-1] = [1, 2, 5, 6, 4]$

- Salida óptima: $\mathbf{s}[0 : k-1] = [4, 2, 3, -1, -1, -1, -1, 1]$

Solución:

Aplicar la siguiente estrategia:

1. Calcular el número total de días de los que se dispone
2. Coger la huerta de máximo beneficio y asignársela al día que le corresponde según su caducidad
3. Coger la siguiente huerta de máximo beneficio y asignársela al día que le corresponde. Si ese día está ocupado, asignársela al día anterior y seguir así hasta poder asignarla o rechazarla
4. Seguir con el paso 3 hasta completar los días.

```
1  public class Hortelano
2  {
3     private static int N=5; //numero de huertas
4     private static int K=8;  //máxima caducidad
5     private static int LIBRE=-1;  //marca que no hay huerta asignada
6
7     private int seleccionCandidato(int h[][]){
8         int j=0;
9
10        for (int i=0;i<N;i++){
11         if (h[1][i]>h[1][j])
12             j=i;
13        }
14        return j;
15    }
16
17    private void quitarCandidato(int h[][], int c){
18                h[1][c]=-1;  //-1 marca de no hay candidato
19    }
20
21    public int [] soluVoraz(int h[][]){
22
23       int s[] = new int [K];
24       int c,j;
25       //inicializa solucion
26       for (int i=0; i<K; i++){
27           s[i]=LIBRE;
28       }
29
30       for (int i=0; i<N; i++){
31           c=seleccionCandidato(h);
32           j=h[0][c]-1; // el día n se anota en la posición n-1
33           quitarCandidato(h,c);
34           while ((j>0) && (s[j]!=LIBRE)){
35             j=j-1;
36             }
37           if ((j>=0)&& (s[j]==LIBRE))
38             s[j]=c;
39
40     }
41     return s;
42 }
```

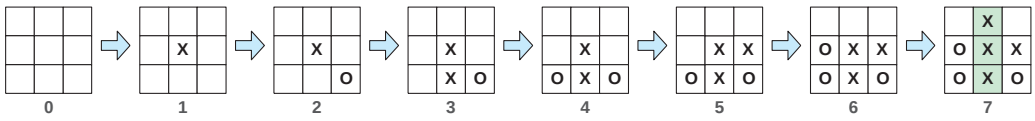
```
43
44 public static void main(String args[]){
45     int huertas[][]={{3,8,2,3,1},{1,2,5,6,4}}; //huertas[0]=fechas caducidad; huertas[1]=beneficios
46     int solucion[];
47
48     hortelano horte = new hortelano();
49     //imprimir datos
50     System.out.print(" h: ");
51     for (int i=0; i<N;i++){
52         System.out.print(i+", ");
53     }
54     System.out.print("\nfc: ");
55     for (int i=0; i<N;i++){
56         System.out.print(huertas[0][i]+", ");
57     }
58     System.out.print("\n b: ");
59     for (int i=0; i<N;i++){
60         System.out.print(huertas[1][i]+", ");
61     }
62     System.out.print("\n");
63
64     //solucion
65     solucion=horte.soluVoraz(huertas);
66     for (int i=0; i<K;i++){
67         System.out.print(solucion[i]+", ");
68     }
69 }
70
71 }
```

Ejercicio 2 – Soluciones del tres en raya [1.5 puntos]

Este ejercicio está relacionado con el juego de las 3 en raya, en el que dos jugadores ubican fichas en un tablero de 3×3 en sucesivos turnos hasta que uno consigue formar una columna, fila, o diagonal con tres fichas suyas (en cuyo caso habría ganado), o se llega a un tablero completo sin ganador (empate).

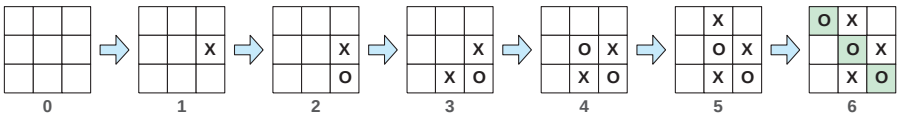
En concreto, se trata de averiguar cuántas veces gana el primer jugador, el segundo, o se empata, considerando todas las posibles secuencias de movimientos por parte de los jugadores. Se pide implementar un programa basado en **backtracking** que genere todas estas posibles secuencias y calcule el número de veces que gana el jugador que pone la primera ficha, la segunda, o se empata. En la búsqueda de soluciones parciales el algoritmo debe parar cuando un jugador haya ganado, o cuando el tablero esté lleno. A continuación se ilustran tres posibles secuencias de movimientos:

- Gana el primer jugador (con ficha X):



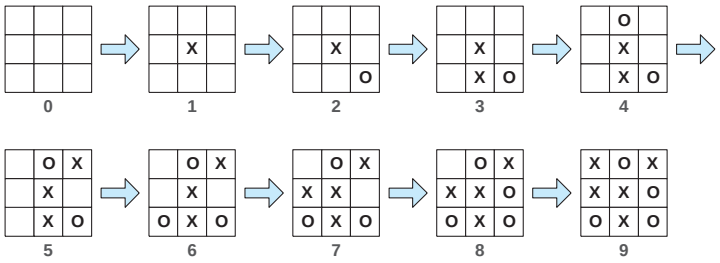
  En el paso 5 el jugador X podía haber colocado una ficha para ganar la partida. De todas formas, el programa no verificará si es posible ganar haciendo movimientos "inteligentes", sino que buscará todas las posibilidades (entre las que se encuentra la ilustrada, continuando en el paso 6).

- Gana el segundo jugador (con ficha O):



  En cuanto un jugador ha ganado ya se dejan de explorar más posibilidades (ya no tiene sentido seguir poniendo más fichas en el tablero).

- Empate:



  En el paso 8 ya se sabe que va a haber un empate. No obstante, solo se contabilizarán empates cuando se hayan rellenado todas las casillas del tablero.

Solución:

```
import java.io.*;

public class TresEnRaya
{
   public static void main(String[] args) throws Exception{

      int[] sols = new int[3]; // almacena el número de veces que se empata, gana, o pierde
      int[] tablero  = new int[3*3]; // versión unidimensional del tablero

      int[] perm = new int[3*3]; // vector de permutaciones
      boolean[] libres  = new boolean[3*3]; // vector auxiliar de elementos libres

      for (int i=0; i<3*3; i++)
         libres[i] = true;

      perms (3*3, 0, perm, libres, sols, tablero);

      System.out.println("Empates: " + sols[0]);
      System.out.println("Gana 1o: " + sols[1]);
      System.out.println("Gana 2o: " + sols[2]);
   }

   private static boolean hay_ganador(int[] t) {

      if (((t[0]==t[1]) && (t[1]==t[2]) && (t[1]!=0)) ||
         ((t[3]==t[4]) && (t[4]==t[5]) && (t[4]!=0)) ||
         ((t[6]==t[7]) && (t[7]==t[8]) && (t[7]!=0)) ||
         ((t[0]==t[3]) && (t[3]==t[6]) && (t[3]!=0)) ||
         ((t[1]==t[4]) && (t[4]==t[7]) && (t[4]!=0)) ||
         ((t[2]==t[5]) && (t[5]==t[8]) && (t[5]!=0)) ||
         ((t[0]==t[4]) && (t[4]==t[8]) && (t[4]!=0)) ||
         ((t[2]==t[4]) && (t[4]==t[6]) && (t[4]!=0)))
         return true;
      else
         return false;
   }

   private static void perms(int n, int i, int[] solucion, boolean[] xs, int[] n_ganadores, int[] tablero){

      int p = (i%2) + 1; // índice del jugador que coloca una ficha

      for (int k=0; k<n; k++)
         if (xs[k]){
            // solucion[i] = k; // no necesario al usar "tablero"
            xs[k] = false;
            tablero[k] = p;

            if (hay_ganador(tablero))
               n_ganadores[p]++;
            else
               if (i==n-1)
                  n_ganadores[0]++;
               else
                  perms(n, i+1, solucion, xs, n_ganadores, tablero);

            xs[k] = true;
            tablero[k] = 0;
         }
   }
}
```

Esta solución sigue el esquema para generar permutaciones. El resultado es:

```
Empates: 46080
Gana 1°: 131184
Gana 2°: 77904
```